(4)

INTEGRATING SYNTAX, SEMANTICS, AND DISCOURSE
DARPA NATURAL LANGUAGE UNDERSTANDING PROGRAM

R&D STATUS REPORT
Unisys/Defense Systems

ARPA ORDER NUMBER: 5262
PROGRAM CODE NO. NR 049-602 dated 10 August 1984 (433)
CONTRACTOR: Unisys Defense Systems
CONTRACT AMOUNT: $1,704,901
CONTRACT NO: N00014-85-C-0012
EFFECTIVE DATE OF CONTRACT: 4/29/85
EXPIRATION DATE OF CONTRACT: 4/28/89
PRINCIPAL INVESTIGATOR: Dr. Lynette Hirschman
PHONE NO. (215) 648-7554

SHORT TITLE OF WORK: DARPA Natural Language Understanding Program

REPORTING PERIOD: - 5/1/88 - 8/1/88

1. Description of Progress

Progress has been severely hampered by the funding situation. Our current funding authorization ran out at the end of June. Since then we have been authorised by Unisys to proceed at risk to a very limited extent, which supports little more than administrative tasks.

1.1. Grammar

Revisions to the grammar have been made to expand coverage of various complex object types and to ensure correct labeling of nodes for semantics in these objects.

1.2. Syntax/Semantics Interaction

We implemented a new version of syntax/semantics interaction that uses the existing semantic interpreter and applies selection restrictions expressed in terms of constraints on thematic role assignments. Use of semantic information associated with thematic roles increases generality and prevents potentially redundant (or conflicting) information from being stored, once as a syntactically based pattern and again as constraints on fillers for the thematic roles associated with a given predicate.

The new mechanism relies heavily on the original SPQR component, and uses the same context free grammar to analyse the ISR. The main difference is that, where before SPQR would simply look the stripped down ISR pattern up in a database, the new mechanism actually runs the semantic interpreter to see if the stripped down ISR is semantically coherent. This has been tested thoroughly on the CASREPS domain, and selects the same parses that SPQR did, in less time. There were a few SPQR patterns that reflected semantic information that could only be provided by time analysis, such as the fact that
[pressure during engagement] is a bad pattern. These are still basically preserved as patterns that are consulted after the semantic interpeter has run. Passing information back from the time analysis component is left as a future research task. Given that qualification, we no longer need to collect a separate set of syntactically-based patterns, although we are planning to preserve a record of ISRs that fail semantics. This will be useful as a source of information about the parser pursuing wrong paths. In order to preserve the knowledge acquisition functionality of SPQR, we are also working on a semantic rule editor which will be called interactively during the parse.

The selection DCG which is used to analyse the ISR has been made almost completely deterministic, yielding a more efficient and elegant module. In addition, another selection switch was implemented, allowing the user to turn on and off the generalization feature of selection, which had previously always been on.

1.3. Semantics

The semantic interpreter was extended to analyze adjectives as prenominal modifiers in the same manner that it interprets prenominal past participles. Some limitations and problems in this extension were documented.

Several meetings were held to discuss the form of the Integrated Discourse Representation (IDR), semantic representations and aspectual operators with the goal of establishing concrete criteria for designing semantic representations for interpreting relations in the IDR. Several meetings were also held to compare the level of functional representation in Lexical Functional Grammar to the ISR.

Routine modifications were made in processing messages from the CASREPS domain which regularized the output of previously working messages and added new messages.

MPACK, the version of KPACK supporting multiple inheritance was installed on the Suns. A Trident knowledge base using MPACK has been implemented. A draft of a report on the MPACK knowledge base has been prepared and is under revision.

1.4. Time

The temporal relations specifying the temporal structure of situations and the partial ordering among them consist of binary Prolog terms. A graphic display of these relations was integrated into the X-window interface. It consists of a header display field with a key explaining the graphic symbols, and a set of windows, each displaying a totally ordered subset. The graphical temporal display has been completed and installed in the stable system.

1.5. Discourse

Processing of the prompt/response relationship has been implemented. A prompt such as *Cause of failure?* creates a context for the proper interpretation of a fragmentary response like *Deterioration due to age and wear,* as being the cause of failure requested in the prompt. We have generalised our treatment of reference to situations so that PUNDIT can recognise references such as *Miller engaged Barsuk. Attack successful,* where the engagement and the attack are the same situation.

1.6. Evaluation of NL systems

At the Darpa Workshop in Mohonk, Martha Palmer organized a panel discussion on the topic of evaluation of natural language processing systems. Several issues were discussed, including the use of training sets and test sets and how they could be established, and the organisation of the next MUCK conference. The week following Mohonk, several of the panel participants, including Martha Palmer, were invited to a meeting at the University of Pennsylvania organized by Mitch Marcus to discuss a possible DARPA proposal that Penn is considering. Penn proposed that several hundred thousand words of data, both written language and spoken language, be collected together and annotated with appropriate syntactic labels to be used as a training set. Part of this data would be kept aside to be used as a hidden test set. Each year more data would be collected to be the test set, and the previous year's test set would be released. Following this meeting, Martha Palmer organized a meeting during the ACL conference where members of the ACL executive committee discussed the issue of evaluation with several DARPA contractors. It was agreed that Penn should hold another meeting to disuss the training set proposal again, and that Martha Palmer should organise a workshop on Evaluation of Natural Language Processing Systems. This workshop is being organized, will be held in December, and is being sponsored jointly by RADC and ACL.

1.7. Documentation

Three pieces of documentation were prepared describing how to use the PUNDIT system:

(1) PUNDIT User's Guide (C. Ball, J. Dowding, F. Lang, C. Weir): a guide to running the PUNDIT system, for the computational linguist familiar with Prolog. Includes appendices documenting PUNDIT files, dependencies, image-building procedures, and references.

(2) A Guide to the PUNDIT Lexical Entry Procedure (L. Riley): documents the procedure for adding new words to the PUNDIT lexicon.

(3) Guide to Object Options in PUNDIT (M. Linebarger): designed to accompany the Guide to the PUNDIT Lexical Entry Procedure. Documents the options available for specifying the allowable objects of verbs in the PUNDIT lexicon.

These documents are included as appendices to this report.

2. Change in Key Personnel

Leslie Riley resigned effective 6/30 to pursue her education.

3. Summary of Substantive Information from Meetings and Conferences

3.1. Darpa Meetings

Shirley Steele, Martha Palmer, and Lynette Hirschman attended the meeting of the Darpa Natural Language contractors at Mohonk, New York, May 4-6.

Lynette Hirschman attended the meeting of Darpa Speech Contractors at Carnegie Mellon University/Hidden Valley June 15-17.

3.2. Papers and Presentations

(1) Hirschman, L. "A Meta-Treatment of wh-Constructions". Presented at the META 88 Workshop on Meta Programming in Logic Programming.

(2) Hirschman, L., Hopkins, W.C., Smith, R.C. "Or-Parallel Speed-up in Natural Language Processing: A Case Study". To be presented at the 5th International Logic Programming Conference, Seattle, August, 1988.

(3) Linebarger, Marcia, Dahl, Deborah, Hirschman, Lynette, and Passoneau, Rebecca, "Sentence Fragments Regular Structures". Presented at the 26th Annual Meeting of the Association for Computational Linguistics", June 6-10, 1988.

(4) Martha Palmer, Lynette Hirschman, and Deborah Dahl, "Text Processing Systems". Tutorial presented at the 26th Annual Meeting of the Association for Computational Linguistics, June 6-10, 1988.

3.3. Conference Attendance

Deborah Dahl, John Dowding, Lynette Hirschman, Martha Palmer, Rebecca Passonneau, and Carl Wier attended the 26th Annual Meeting of the Association for Computational Linguistics at Buffalo, New York, June 6-10. Pundit was demonstrated at this conference.

Lynette Hirschman attended the Meta88 conference on meta-programming in logic programming, Bristol, England, in June.

Lynette Hirschman attended the 5th International Logic Programming Conference, Seattle, August, 1988.

4. Problems Expected or Anticipated

Authorized funding ran out in June. We have been informed that an additional $86K has been signed off out of Darpa, and that the $412K increment is being processed as well, but that these will probably not be released from ONR until October. We cannot resume work until this funding is received. In addition, it is critical that we receive our FY 89 funding as soon as possible; otherwise it will be necessary to interrupt work again, pending receipt of that funding.

5. Action Required by the Government

Expedite $86,000 FY88 funding increment. Expedite FY89 funding.

6. Fiscal Status

(1) Amount currently provided on contract:

$ 1,192,833 (funded) $1,704,901 (contract value)

(2) Expenditures and commitments to date:
$ 1,217,835

(3) Funds required to complete work:
$ 487,068

PUNDIT
User's Guide*

Version 1.0

July 6, 1988

Unisys Logic-Based Systems
Paoli Research Center
P.O. Box 517, Paoli, PA 19301

*This work has been supported by DARPA contract N00014-85-C-0012, administered by the Office of Naval Research.

Contents

List of Figures

1 Introduction

1.1 The *User's Guide*

The *PUNDIT User's Guide* is intended to provide a concise and general introduction to the facilities of the PUNDIT text-processing system. The intended audience is computational linguists familiar with Quintus Prolog. While this document is not a reference manual, and does not in itself contain sufficient information for you to either extend the system or port it to a new domain, we have tried to cover the operational basics: how to run PUNDIT (Section 2) and how to interpret PUNDIT's output (Section 3). In addition, Section 4 documents the two main procedures for accessing the system (**parse** and **pundit**), as well as a number of other procedures which we make frequent use of as developers. Appendix A and Appendix B will help you set the system up. Appendix D identifies the core and domain files, and Appendix E lists papers, presentations, and technical documentation available for PUNDIT.

1.2 The Software

The *User's Guide* is designed to accompany a subset of the text-understanding software which has been developed at the Paoli Research Center, as it exists on the date of publication: the core components of PUNDIT, together with the domain-specific components developed to process Navy tactical messages (RAINFORMS). This domain will be referred to henceforth as the MUCK domain (an acronym for the message understanding conference which occasioned the development of the software). The MUCK software is essentially similar to that developed for other domains, and may be considered representative: it includes a domain-specific message input screen, lexicon, knowledge base, semantics rules and database definitions, and it supports both analysis of text and limited natural language queries. It differs from other domain software chiefly in having a comparatively rich knowledge base.

2 Running PUNDIT

2.1 Core Images and Domain Images

Before you can use PUNDIT, the software must be installed at your site and the images built. Appendix A contains instructions for creating a PUNDIT core image and a MUCK domain image.

The core image is not functional, and is generally used only to build the domain images.[1] In the discussion that follows, it will be assumed that you have a MUCK domain image available to you.

2.2 The MUCK Domain

The MUCK domain has been designed to process the *Remarks* field of Navy tactical messages. Since the formatted fields in these messages contain information which establishes the initial context for interpreting the text (message originator, date/time, etc.), we have developed a special front-end to collect this information. This message front-end is accessed by issuing the command **pundit**. See Section 4 for more information about this command.

In order to make use of the MUCK domain image for syntactic and semantic analysis of natural language input, you will need to know something about the sublanguage and the knowledge base for this domain. In the file **muck_working.pl** you will find a subset of the messages from our message corpus which PUNDIT is currently able to process. By examining other domain-specific files such as the lexicon, the knowledge base, and the semantics rules, you should be in a position to construct your own input (see Appendix D for a list of these files).

2.3 parse and pundit

The **pundit** command (discussed above) invokes the domain-specific message processing front-end to the system, which collects both message header information and the message body. An alternative, domain-independent method of accessing the system is provided by **parse**, which prompts only for the text to be processed. Many of the researchers working on PUNDIT currently interact with the system using **parse**, although certain higher-level processes—reference resolution in particular—do not perform as well as they otherwise could, since the initial discourse context is empty. The **parse** command, however, provides more options for developers, and is the only command to use when no semantic processing is desired (the front-end invoked by **pundit** assumes that a complete analysis is required). These two commands are discussed in more detail in Section 4.

[1] The core image contains only the core procedures of PUNDIT, including the core lexicon (see Appendix D). See Appendix B for details on how to create a functional image from the core image.

2.4 Before You Begin

Since we will be using a text from the MUCK domain to illustrate PUNDIT's operation, at this point you may wish to load the MUCK image. Before using **parse** or **pundit**, however, you will first need to set a few of the software switches which enable or disable various system features. Do this by executing the **switches** procedure (described in more detail in Section 4). The **switches** procedure will display the current switch settings in the image, and will prompt you for a list of switches to be changed. Make sure, at least for now, that you have the following switches turned on, and that all the others are turned off:

1. **parse_tree**
2. **conjunction**
3. **semantics**
4. **translated_grammar_present**
5. **translated_grammar_in_use**
6. **selection**

At this stage you may also want to tell the Selection module not to query you about new co-occurrence patterns. Call the procedure **ssucceed** (see Section 4 for more details).

2.5 Processing a Sentence

Having brought up the MUCK domain image and set your switches, you are are now ready to analyze a sentence. Call **parse**, and you should see the prompt "**sentence:**". Since the following section describes the output generated from processing the sentence *visual sighting of periscope followed by attack with asroc and torpedos.*, you might want to type it in now, including the final period. After typing the sentence in, you will need to signal the end of input by entering **two carriage-returns**. The following is a transcript of someone doing what you have just been asked to do in the last two subsections[2].

[2]Note that if you later create a **prolog.ini** file, as described in Appendix C, your initial switch settings may differ from those shown in the figure.

```
%/nlp/nlp/pundit/muck/Muck.qimage

Quintus Prolog Release 2.2 (Sun-3, Unix 3.2)
Copyright (C) 1987, Quintus Computer Systems, Inc.  All rights reserved.
1310 Villa Street, Mountain View, California  (415) 965-7700

| ?- switches.

1. enter_new_word------------------> OFF
2. np_trace------------------------> OFF
3. parse_tree----------------------> OFF
4. conjunction---------------------------------> ON
5. semantics-----------------------> OFF
6. translated_grammar_present------------------> ON
7. translated_grammar_in_use-------> OFF
8. grinder-------------------------> OFF
9. text_mode-----------------------> OFF
10. decomposition_trace------------> OFF
11. summary------------------------> OFF
12. show_isr-----------------------> OFF
13. selection----------------------------------> ON
14. enable_db_access---------------> OFF
15. count--------------------------> OFF
16. all_time-----------------------> OFF
17. time_trace---------------------> OFF
18. window_display-----------------> OFF
Please choose a list of switches, or type "ok."  -- [3,5,7].

Changed the switch: parse_tree-----------------------------------> ON
Changed the switch: semantics------------------------------------> ON
Changed the switch: translated_grammar_in_use--------------------> ON
yes
| ?- ssucceed.
Setting selection switch unknown_selection to -----------> succeed

yes
| ?- parse.

sentence: visual sighting of periscope followed by attack with asroc
 and torpedos.
```

Figure 1: Running PUNDIT

3 Interpreting PUNDIT Output

Syntactic processing in PUNDIT yields two syntactic descriptions of a sentence: a detailed surface structure parse tree, and an operator-argument representation called the *Intermediate Syntactic Representation*, or ISR. The ISR regularizes the information in the parse tree, reducing surface structure variants to a single canonical form and eliminating details not required for semantic analysis.

PUNDIT's semantic and pragmatic components take the ISR as input and produce a final representation of the information conveyed by the sentence which includes a decomposition of verbs into a structure of more basic predications, resolution of anaphoric references, and an analysis of temporal relations. The resulting data structure is known as the *Integrated Discourse Representation*, or IDR.

These three kinds of output will be illustrated for the following sentence:

Visual sighting of periscope followed by attack with asroc and torpedos.

This particular sentence is characteristic of the sort of input PUNDIT has been designed to handle. Note the ellipsis typical of message sublanguages[3].

3.1 The Parse Tree

The syntactic analyses produced by PUNDIT are in the formalism of String Grammar [Sager 81]. A brief glossary of String Grammar terms is provided below in figure (2) for help in understanding the parse tree in figure (3). Parse trees are displayed with siblings indented to the same depth; terminal elements (lexical items) are preceded by ==.

3.2 The ISR

The ISR corresponding to the parse tree in figure (3) is shown in figure (4), which is taken from the output of the **parse** procedure. Two versions of the ISR are given: the first is essentially the data structure passed to semantic analysis, and the second is a pretty-printed version.

The ISR requires little knowledge of string grammar to understand. Each clause consists of syntactic operators (OPS—generally tense and aspect markers derived from the verb morphology), the verb or predicate (VERB), and its arguments. Conjunction is indicated by the insertion of the conjunction, followed by the conjuncts (set off by parallel lines). Note that each noun phrase has an associated referential index; in this example, the ISR has been printed after semantic and pragmatic analysis, and the indices have been bound to discourse entities (`[sight1]`, `[periscope1]`, etc.).

[3]Translation: The visual sighting of a periscope was followed by an attack (on the submarine) with anti-submarine rockets and torpedos.

`lxr`	$\Longleftrightarrow$	a left-adjunct + x + right-adjunct construction, where x can be:
		`n` $\Leftrightarrow$ a common noun
		`a` $\Leftrightarrow$ an adjective
		`v` $\Leftrightarrow$ a verb
		`ven` $\Leftrightarrow$ a past participle
		`tv` $\Leftrightarrow$ a tensed verb
		`ving` $\Leftrightarrow$ a present participle
		`q` $\Leftrightarrow$ a quantity word
		`pro` $\Leftrightarrow$ a pronoun
`nstgo`	$\Longleftrightarrow$	noun string object
`nstg`	$\Longleftrightarrow$	noun string
`sa`	$\Longleftrightarrow$	sentence adjunct
`pn`	$\Longleftrightarrow$	preposition + noun (prepositional phrase)
`tpos`	$\Longleftrightarrow$	the/determiner (prenominal) position
`qpos`	$\Longleftrightarrow$	quantity (prenominal) position
`apos`	$\Longleftrightarrow$	adjective (prenominal) position
`npos`	$\Longleftrightarrow$	noun (prenominal) position
`venpass`	$\Longleftrightarrow$	past participle + passive
`passobj`	$\Longleftrightarrow$	passive object
`nullobj`	$\Longleftrightarrow$	null object (for intransitive verb)
`thats`	$\Longleftrightarrow$	that + sentence object
`objbe`	$\Longleftrightarrow$	object of be
`vingo`	$\Longleftrightarrow$	present participle + object
`commaopt`	$\Longleftrightarrow$	comma option
`conj_wd`	$\Longleftrightarrow$	conjunction word
`spword`	$\Longleftrightarrow$	special (conjunction) word
`dstg`	$\Longleftrightarrow$	adverb string, where `d` stands for adverb.

Figure 2: A glossary of string-grammar terms

```
sentence
 center
  fragment
   zerocopula
    subject
     nstg
      lnr
       ln
        apos
         adjadj
          larl
           avar
            adj == visual
        nvar
         ving == sighting
        rn
         pnpn
          pn
           p == of
           nstg
            lnr
             ln
             nvar
              n == periscope
    lvr
     vvar
      null`aux
    object
     be`aux
      venpass
       lvenr
        ven == followed
        sa
         pn
          p == by
          nstg
           lnr
            ln
            nvar
             n == attack
            rn
             pnpn
              pn
               p == with
               nstg
                lnr
                 ln
                 nvar
                  n == asroc
                 conj`wd
                  spword == and
                 lnr
                  ln
                   tpos == tagged local
                   qpos == tagged local
                   apos == tagged local
                   npos == tagged local
                  nvar
                   n == torpedos
        passobj
         nullobj
 . == .
```

Figure 3: Parse tree for *Visual sighting of periscope followed by attack with asroc and torpedos.*

```
INTERMEDIATE SYNTACTIC REPRESENTATION (ISR):

[untensed,follow,subj(passive),obj([tpos(□),[gerund,nvar([sight,singular,
[sight1]]),pp([of,[tpos(□),[nvar([periscope,singular,[periscope1]])]]])],
adj([visual])]),pp([by,[tpos(□),[nvar([attack,singular,[attack1]]),pp([with,
[and,[tpos(□),[nvar([anti^submarine^rocket,singular,[rocket1]])]],[tpos(□),
[nvar([torpedo,plural,[torpedos1]])]]]]])]]])]

OPS:   untensed
VERB:  follow
SUBJ:  passive
OBJ:   gerund: sight (sing) : [sight1]
          L_MOD:  adj: visual
          R_MOD:  pp: of
                      periscope (sing) : [periscope1]
PP:    by
       attack (sing) : [attack1]
          R_MOD:  pp: with
                              and
                      ------------------------------
                      anti^submarine^rocket (sing) : [rocket1]
                      ------------------------------
                      torpedo (pl) : [torpedos1]
```

Figure 4: ISR for *Visual sighting of periscope followed by attack with asroc and torpedos.*

3.3 The IDR

The IDR for the example sentence is shown in figure (5); its major segments are labelled **Ids**, **Properties**, **Events and Processes**, **States**, and **Important Time Relations**.

The **Ids** segment lists all the **id**, **is_group**, and **generic** predications derived during the analysis of the example sentence. **Generic** relations are established primarily to support subsequent reference through generic *they* or *one*-anaphora[4]. **Id** relations indicate the semantic type of each non-group discourse entity, while the **is_group** relations specify the semantic type, members, and cardinality of each group-level discourse entity. Thus for example the **id** relation for the entity **[sight1]**[5], derived from the nominalization *visual sighting of periscope*, indicates that the entity is an **event**, while the **is_group** relation for the entity **[projectiles1]** indicates that the entity is a group of projectiles, consisting of an unknown number of rockets and torpedos.

Relations in the **Properties** segment of the IDR are heterogeneous: these are miscellaneous relations derived in the course of processing noun phrases. Prenominal adjectives typically give rise to such relations; processing of noun-noun compounds may generate **unspecified_relationship** predications if no relationship between the nouns can be derived from domain knowledge. In the current example, the **reportingPlatform** relations are generated by a procedure which creates a default entity if the identity of the message originator is not known—if we had used the **pundit** procedure instead of **parse**, this information would have been supplied by the message header.

The **Events and Processes** and **States** segments of the IDR contain predications over discourse entities which denote situations[6]. Typically it is the processing of a clause or a nominalization which gives rise to a situation entity, and if the situation is an event, then an entity will be generated for the resulting state as well. The main predicate is the type of situation (event, state, or process), and each predication has three arguments:

1. The discourse entity
2. The associated semantic representation
3. A moment or period of time for which the situation holds

For example, the first predication in the **Events and Processes** segment in figure (5) was derived from processing the ISR for the nominalization *visual sighting of periscope*. This particular predication asserts that the referent introduced by the gerund *sighting* denotes an event; the semantic representation was constructed based on the semantics rules for the verb *sight*. All situations that are labelled events in PUNDIT can be more

[4]See [Dahl 84] for a description of the relationship between generics and one-anaphora.

[5]Labels for discourse entities are derived from the lexical head of the expression and are typically enclosed in brackets. These labels are arbitrary; **[entity2]** would do equally well.

[6]See [Passonneau 87] for a more detailed discussion of the semantics of situations.

accurately described as transitions from one state into another, where the full temporal structure of the event consists of an initial process interval, the moment of transition, and the new situation that is entered into[7]. In the second argument of the predication, the **becomeP** operator takes as its argument the semantic representation that gives rise to the new situation that is entered into, **[sight2]**. The third argument of the predication, **moment([sight1])**, should be interpreted functionally as returning the moment at which the transition into the state in question occurred. Information about this new state, **[sight2]**, is provided by a predication in the **States** field.

The final segment of the IDR lists the temporal relations which were analyzed as holding amongst the situations. Note in particular that since the verb *follow* is defined as a temporal operator, PUNDIT has correctly established the temporal relationship between the sighting and the attack.

[7]There is no referent introduced for the initial process interval of transition events.

```
Ids:
generic(torpedo)
is_group([torpedos1],members(torpedo,[torpedos1]),numb(_21227))
generic(anti^submarine^rocket)
id(anti^submarine^rocket,[rocket1])
is_group([projectiles1],members(projectile,[[rocket1],[torpedos1]]),numb(_21279))
id(us_platform,[us_platform1])
id(process,[attack1])
generic(periscope)
id(periscope,[periscope1])
id(us_platform,[us_platform3])
id(state,[sight2])
id(event,[sight1])

Properties:
reportingPlatform([us_platform1])
reportingPlatform([us_platform3])

Events and Processes:
event(
  [sight1]
  becomeP(sightP(experiencer([us_platform3]),theme([periscope1]),instrument(visual)))
    sighted_atP(theme([periscope1]),location(_28507))
  moment([sight1]))

process(
  [attack1]
  doP(attackP(actor([us_platform1]),theme(_19607),instrument([projectiles1])))
  period([attack1]))

States:
state(
  [sight2]
  sightP(experiencer([us_platform3]),theme([periscope1]),instrument(visual))
    sighted_atP(theme([periscope1]),location(_28507))
  period([sight2]))

Important Time Relations:
the sight state ([sight2]) started with the sight event ([sight1])
the sight event ([sight1]) preceded the arbitrary event time (moment([attack1]))
of the attack process ([attack1])
```

Figure 5: IDR for *Visual sighting of periscope followed by attack with asroc and torpedos.*

4 Commonly Used Procedures

4.1 edit_rule

The procedure **edit_rule/1** allows you to edit a set of grammar rules for a specified non-terminal, using the Prolog Structure Editor. For more details, please consult [Riley 86].

4.2 edit_word

The procedure **edit_word/1** allows you to edit the lexical entry for a specified word, using the Prolog Structure Editor. For more details, please consult [Riley 86].

4.3 parse

The procedures **parse** and **pundit** (see below) provide two slightly different front-ends to the PUNDIT system. **parse** is the access method of preference for those whose primary interest is parsing or minimizing keystrokes (no prompts are issued to collect message header information). The **parse** procedure is a core component of PUNDIT, and is domain-independent.

The behavior and output of **parse** are largely controlled by switch settings (see Section 4). Briefly, the **parse** procedure collects the input to be analyzed by PUNDIT, and then calls syntactic analysis. Depending on your switch settings, it may then call semantic analysis, the database extractor, and the summary module (if defined for the current domain). Depending again on switch settings, you may be shown both intermediate and final results: trace messages, the parse trees, the ISRs, the IDR, database relations extracted, and a summarization of the input text[8]. In the course of processing your input, PUNDIT may engage you in dialogue if certain switches are turned on: for example, the Selection module may ask you about co-occurrence patterns; if the switch **enter_new_word** is on, you will be prompted to enter lexical information for new words.

The initial prompt to collect the input depends on switch settings as well. If the switch **text_mode** is on, you will be prompted to enter a paragraph of text: that is, one or more sentences followed by two carriage returns[9]. In this case, the input will be processed one sentence at a time, and the first parse for each sentence will be processed.

If the switch **text_mode** is off, you will be prompted to enter a single sentence; after processing the first parse, you will be invited to continue with the next parse, until you wish to stop or all parses have been exhausted.

[8]The summary application is not implemented in the MUCK domain.

[9]Since each sentence may optionally be followed by one carriage return, the extra carriage return at the end is needed to signal the end of input. Moreover, although PUNDIT will process run-on sentences (without punctuation), the final sentence must have a terminator: a period, exclamation point, or question mark.

In addition to these capabilities, designed for the processing of sentences, you may also analyze lower-level constituents. To process an isolated noun phrase, call **parse_np/0** (this procedure supports both syntactic and semantic analysis). NPs and other constituents may also be parsed by invoking **parse/1**, giving as argument the grammatical category (this will require a knowledge of PUNDIT's grammatical categories). As a simple illustration, you may parse the noun phrase *visual sighting of periscope* by calling **parse(lnr)**. Note, however, that **parse(lnr)** does not support semantic analysis.

4.4 pundit

The **pundit** procedure provides a domain-specific front-end to the PUNDIT system, one geared specifically towards full message processing. Since **pundit** is similar in many respects to **parse** (see above), only differences will be described here.

First, **pundit** is not sensitive to the **semantics** and **text_mode** switches: it is assumed that all messages require semantic analysis, and that all input will be one or more sentences of text. As a result, it is not possible to request multiple parses of the input. However, if a sentence fails semantic analysis, **pundit** will backtrack for the next parse, and this process will continue until a semantically acceptable parse is found.

Secondly, **pundit** provides a domain-specific message entry screen which collects the message header and the message body. The screen for the MUCK domain is shown in Figure (6) below (you may enter a question mark at any prompt to receive a description of valid responses). The responses to the first four prompts are used to establish the discourse context for the interpretation of the message body.

The **pundit** procedure also provides capabilities for processing one or more existing messages from the message corpus (stored in **<domain>_working.pl**). When you first invoke **pundit**, the message corpus is compiled into your image, creating entries in the recorded database[10]. At the prompt for **Message number**, you may enter the number of an existing message, and **pundit** will fetch the message from the recorded database and process it. If you wish to process a list of existing messages, call **pundit(batch,YourList)**, where YourList is a Prolog list of message numbers. You may also process the entire message corpus by calling **pundit(batch,test_pundit)**[11].

[10] If there is a version of the message corpus in your directory, **pundit** will load that; otherwise, it will load the file from the main domain directory. This feature allows you to maintain a personal corpus of texts.

[11] This is the method which we use to test software changes: the output can be saved in a file and compared against the results of testing a previous image.

```
%~nlp/pundit/muck/Muck.qimage +

Loading /usr/local/bin/em215 with /mn2/q2.2/ml...
Unix Prolog+Emacs V2.15 (01-Jan-88)
Copyright(c) 1986, 1987 Unipress Software, Inc.

Quintus Prolog Release 2.2 (Sun-3, Unix 3.2)
Copyright (C) 1987, Quintus Computer Systems, Inc.  All rights reserved.
1310 Villa Street, Mountain View, California  (415) 965-7700

 [consulting /mn2/cball/prolog.ini...]
Setting selection switch unknown_selection to ------------> succeed
 [prolog.ini consulted 0.133 sec 720 bytes]

| ?- pundit.

[compiling /nlp/nlp/pundit/muck/muck_working.pl...]
[muck_working.pl compiled 2.700 sec 12,612 bytes]

************************* RAINFORM MESSAGE ENTRY **********************

Message number             [1] :11
Enemy platform       [barsuk] :submarine
Reporting platform [virginia] :texas
Report time           [0800t] :0800t

Sighting message: sighted periscope an asroc was fired proceeded to
station visual contact lost, constellation helo hovering in vicinity.
sub appeared to be ooa.

     Processing discourse segment...

..

     Segment processing Time: 39.967 sec.


****************************** Complete IDR ****************************

(etc.)
```

Figure 6: Using the pundit procedure

4.5 punt

This procedure provides on-line documentation for several PUNDIT utilities: the *Prolog Structure Editor*, the *Lexical Entry Procedure*, tools for creating a concordance, and the *Dictionary Merge* utility. To invoke the punt utility, type punt at the Prolog prompt.

4.6 rdb_remove

This development utility removes entries of specified type(s) from the Prolog recorded database. It is useful when testing changes to one of the files whose compilation creates such entries. For example, the pundit procedure, as one of its steps, compiles the message corpus into your current image. If you should wish to edit and reload the message file (<domain>_working.pl), you must first remove the old messages: rdb_remove facilitates this task. A sample session is given below.

```
| ?- rdb_remove.

Recorded Database Rules:
1. The Lexicon (dict)
2. The Bnf (bnf)
3. Define and Simplification Rules (define)      [obsolete]
4. Semantic Selection Rules (semantics)          [obsolete]
5. Clause Mapping Rules (mapping)                [obsolete]
6. Noun Phrase Mapping Rules (mapping_np)        [obsolete]
7. All Semantics Rules (all_semantics)           [obsolete]
8. The Selectional Patterns (selection)
9. The Stable Messages (messages)
10. quit
Please choose a list of items --  [9].

Erasing corpus muck...

Time to erase the testing messages: 0.15 sec.
```

Figure 7: Using the rdb_remove utility

Note that options 3-7 are obsolete (semantics rules are not stored in the recorded database).

4.7 readIn

The procedure **readIn/1** loads a PUNDIT lexicon into the current image. Its argument is the name of a lexicon file. For example, to load the lexicon file **my_lex.pl** from the current working directory, execute the goal **readIn(my_lex)**. Lexical entries are stored in the recorded database; to avoid duplicate entries, it may be necessary to run **rdb_remove** to remove previous entries before using **readIn** to load a new lexicon.

4.8 squery

The predicate **squery/0** is used to control the behavior of the Selection component when it encounters an unknown selectional pattern. Execute the goal **squery** to be queried when an unknown pattern is encountered. For more details, see Section 12 of [Lang 87].

4.9 ssucceed

The predicate **ssucceed/0** is analogous to **squery/0**, except that it is used to allow unknown selectional patterns to succeed. There is also a predicate **sfail/0** which can be used to force unknown selectional patterns to fail. For more details, see [Lang 87].

4.10 switches

The **switches** utility allows you to control the operation of PUNDIT. Each switch and its dependencies are described in more detail below.

```
| ?- switches.

1. enter_new_word------------------> OFF
2. np_trace------------------------> OFF
3. parse_tree----------------------> OFF
4. conjunction---------------------------------> ON
5. semantics-----------------------> OFF
6. translated_grammar_present------------------> ON
7. translated_grammar_in_use-------> OFF
8. grinder-------------------------> OFF
9. text_mode-----------------------> OFF
10. decomposition_trace------------> OFF
11. summary------------------------> OFF
12. show_isr-----------------------> OFF
13. selection----------------------------------> ON
14. enable_db_access---------------> OFF
15. count--------------------------> OFF
16. all_time-----------------------> OFF
17. time_trace---------------------> OFF
18. window_display-----------------> OFF
Please choose a list of switches, or type "ok."  -- [5,7,9].


Changed the switch: semantics---------------------------------> ON

Changed the switch: translated_grammar_in_use-----------------> ON

Changed the switch: text_mode---------------------------------> ON
```

Figure 8: Using the **switches** utility

Several related procedures are useful in this connection. The procedure **status** displays current switch settings; `flip/1` reverses the setting of one switch (for example, `flip(semantics)`); `turn_on/1` and `turn_off/1` turn a specified switch on and off.

4.10.1 enter_new_word

This switch controls the behavior of PUNDIT when lexical lookup encounters a word which is not in the lexicon and which cannot be analyzed by the Shapes module. If the input to PUNDIT contains an unrecognizable word and this switch is **off**, lexical lookup will issue the following error message:

`No definition found for --` *<UNKNOWN-WORD>*

`sentence failed ...`

If the switch is **on**,you will be given the following options:

1. Respell word
2. Add dictionary entry
3. Word is a proper noun
4. Quit

Choose the first option if you have simply misspelled the word. If the word is a proper name, you may choose the third option (but no dictionary entry will be created). If you choose to add a new dictionary entry, the *Lexical Entry Procedure* is invoked, and you will be prompted to enter morphological and grammatical information, which may be optionally saved in a file in your directory (consult [Riley 88] and [Linebarger 88] for more detail). Note that the information collected will allow PUNDIT to proceed with the syntactic analysis of the input, but may not be sufficient to enable semantic analysis: for this, it may be necessary to add new semantics rules and/or update the knowledge base.

4.10.2 np_trace

This switch controls the display of Reference Resolution trace messages concerning the creation of discourse entities. Turning this switch on will only have an observable effect if the `semantics` switch is turned on as well.

4.10.3 parse_tree

This switch controls printing of the parse tree and the ISR. The parse tree and ISR are always computed whether this switch is on or not.

4.10.4 conjunction

This switch is one of several switches that cannot be switched. The switch will be **on** if the conjunction meta-rule has been applied to the grammar, and will be **off** otherwise. If this switch is off, and you want the grammar to include conjunction, run the procedure **gen_conj/0**. After the meta-rule has been applied, the switch will automatically be turned on. Since the meta-rule cannot be undone, the switch cannot subsequently be turned off.

4.10.5 semantics

Turn this switch **on** to enable semantic and pragmatic analysis of input; turn it **off** if you wish only to parse. Only the **parse** procedure is sensitive to this switch: the **pundit** procedure assumes that you want a full analysis of the input.

4.10.6 translated_grammar_present

The switch indicates whether or not the grammar has been translated into Prolog. The switch is **on** in the software which accompanies this document, and cannot be turned off.

If at your site an image has been developed in which this switch is **off**, then the grammar must be run interpreted. Running interpreted is slow, but it facilitates debugging and rapid grammar changes. Turning the switch on will translate the grammar, which may take a few minutes; after translation, you will be given the option to compile the resulting Prolog code. You will normally want to do this, because the compiled translated grammar provides the fastest parsing. The only reason not to do this is if you want to use the Prolog debugger on the translated code, which is not advised. If at any time you want to compile the translated grammar, compile the file **translated_grammar.pl**.

4.10.7 translated_grammar_in_use

This switch allows you to parse with the grammar translated (**on**) or interpreted (**off**). Although the switch is off in the software which accompanies this document, you will normally want it to be on (for the fastest parsing). The only reason to turn this switch off is to make use of certain grammar debugging tools that are only available when interpreting the grammar, such as **grinding** and **counting**.

4.10.8 grinder

This switch allows you to trace the application of grammar rules and restrictions, a development feature which is only available when parsing with the grammar interpreted (if

you turn this switch **on**, the **translated_grammar_in_use** switch will automatically be turned off).

The facility is called *grinder* because it typically produces considerable output. To reduce the amount of output, you may choose to trace only the application of specific grammar rules or restrictions.

```
| ?- turn_on(grinder).

Enter one of:  [<what you want to grind on>],
               off,  or
               all
 ** WARNING ** If you grind at all, you will automatically run interpreted.
Enter choice:
```

Figure 9: Setting the **grinder** switch

4.10.9 text_mode

This switch is used by the procedure **parse**. If it is **on**, you will be prompted to enter a paragraph of text (one or more sentences followed by two carriage returns). Only the first parse for each sentence in the paragraph will be processed. If the switch is **off**, you will be prompted to enter a single sentence, and you may step through all parses for that sentence.

4.10.10 decomposition_trace

This switch allows you to monitor the course of semantic analysis: if it is **on**, a variety of trace messages will be displayed, including the ISR for each clause about to be processed and the semantic representation of the input as it is built up. While the switch was designed to facilitate development of semantics rules and the knowledge base, the trace messages are also useful when diagnosing the source of an incorrect or unsuccessful semantic analysis. Note that **decomposition_trace** has no effect unless the **semantics** switch is also on.

4.10.11 summary

This switch controls whether or not a domain-specific module is called to create a summary of the input text. Since summaries depend on the output of semantic analysis, the **semantics** switch must be turned on. Note: the summary application has not been implemented in the MUCK domain.

4.10.12 show_isr

This switch controls the display of the ISR; its effect depends on whether you are using **parse** or **pundit**. If the switch is **on** and you are using the **parse** procedure, the incremental ISR will be displayed for each node in the parse tree. This is useful for debugging changes to the ISR, but not recommended otherwise. Note that the **parse_tree** switch must also be on in this case (when using **parse**, you cannot see the ISR without also displaying the parse tree).

If you are using the **pundit** procedure and this switch is **on**, the ISR for each sentence will be displayed after syntactic analysis and before semantic analysis. In this case, the **parse_tree** switch need not be on.

4.10.13 selection

This switch controls whether or not the Selection module is invoked in the course of parsing. If it is **on**, Selection will be called; if it is **off**, Selection will not be called. For more details, see [Lang 87].

4.10.14 enable_db_access

This switch controls whether or not queries and assertions access the database defined for the current domain. It is used by the procedures **parse** and **pundit**. If the switch is **on**, domain-specific database definitions will be used to extract database relations from the results of semantic analysis, and these relations will be displayed on your screen.

Dependencies: **semantics** must be turned **on**, and database relations must be defined for the current domain (<domain>_db_structure.pl and <domain>_db_mapping.pl).

4.10.15 count

This switch should be left off.

4.10.16 all_time

This switch controls the display of the time relations segment of the IDR. If it is **off**, the segment is labelled **Important Time Relations** and contains what are judged to be the most prominent temporal relations discovered during temporal analysis of the input. If it is turned **on**, the segment is labelled **Complete Time Relations**, and all the relations that could be discovered are displayed. Turning this switch on will only have an observable effect if the **semantics** switch is turned on as well.

4.10.17 time_trace

This switch allows you to monitor the course of temporal analysis. If it is **on**, informative trace messages will be displayed about situation representations as they are constructed by the Time component. Turning this switch on will only have an observable effect if the **semantics** switch is turned on as well.

4.10.18 window_display

This switch should be left off.

A Installing the System

The PUNDIT system runs under release 4.3 of Berkeley UNIX and release 2.2 of Quintus Prolog. Before installing PUNDIT, a `/nlp` partition should first be created; this partition should contain the directory `/nlp/nlp/pundit`, where the core PUNDIT components will be installed. Software for the MUCK domain will be installed in the `/nlp/nlp/pundit/muck` subdirectory.

If these partitions and directories cannot be created, several absolute path names in PUNDIT code will require modification: the files and lines of code are listed below. Note that if it is necessary to create alternative directories to those recommended, please ensure that core PUNDIT files and domain-specific files are stored in separate directories.

FILENAME	code
punt.pl	`:- asserta(home_dir("//nlp/nlp/pundit/")).`
qprolog15.pl	`timeCom :- unix(shell('/mn2/AI/nlp/bin/timeCom')).`
sem_edit.pl	`:- compile('~nlp/pundit/semed/correctForms.pl').`
switches.pl	`compile('~nlp/pundit/count_on.pl').`
switches.pl	`compile('~nlp/pundit/count_off.pl').`
compilePundit	`pundit_directory('/nlp/nlp/pundit').`
compileMuck	`muck_directory('/nlp/nlp/pundit/muck').`

We strongly recommend that the files in the PUNDIT home directory (and its subdirectories) be owned by a special user, and that the file protections be set in such a way that only this special user can alter these files.

B Building PUNDIT Images

B.1 Building a Core PUNDIT Image

To create a core PUNDIT image, execute the following sequence of steps:

1. get in a directory to which you have write permission
2. start up Quintus Prolog 2.2
3. compile the file `/nlp/nlp/pundit/compilePundit`

Compiling the `compilePundit` file will deposit in the current working directory a Prolog saved state called `Pundit.testimage`, which is the core PUNDIT image.

B.2 Creating a Functional Core PUNDIT Image

The core PUNDIT image itself is not functional (i.e., it cannot be used to parse sentences), and is only used to build the domain-specific images. If, however, a user wishes to make a functional image from a core PUNDIT image, the following steps should be executed:

- Create a file containing the following Prolog code:

```
% ------------------------------------------------------------

% Turn on conjunction and translate the grammar
:- gen_conj.
:- translate_grammar('/nlp/nlp/pundit/translated_grammar.pl').
:- compile('/nlp/nlp/pundit/translated_grammar.pl').
:- compile('/nlp/nlp/pundit/muck/compute_types.pl').

% These declarations are required for the Selection module
pundit_domain(core).

isa(nothing,nothing).

semantic_type(nothing,nothing).

% ------------------------------------------------------------
```

- Start up the core PUNDIT image and compile the file containing the code above.
- Save the resulting image (e.g., by executing the goal `save_program('Pundit.newimage')`.

Note that this image can be used only for parsing, since most of the procedures required for semantic analysis (e.g. the knowledge base and semantics rules) are domain-specific.

B.3 Creating a Complete Domain-Specific Image

To create a complete domain-specific image (in this case, an image for the MUCK domain), follow these steps:

- Start up the core PUNDIT image. This image must be the basic non-functional core PUNDIT image described in Appendix B.1, and *not* the functional core PUNDIT image described in Appendix B.2.
- Compile the file `/nlp/nlp/pundit/muck/compileMuck`.

At the beginning of the compilation of the file `compileMuck`, the user is asked three questions:

1. `Do you want to turn on conjunction? (y or n)`
2. `Do you want to translate the grammar? (y or n)`
3. `Do you want to compile the translated grammar? (y or n)`

The user will normally answer "`y`" to each of these. Compiling the `compileMuck` file will deposit in the current working directory another Prolog saved state called `Muck.testimage`, which is the complete domain image.

Once the above procedure has been completed, and the user has exited Prolog, either of these two Prolog saved states can be started up simply by typing `Pundit.testimage` or `Muck.testimage` to the UNIX prompt (or by typing the absolute filename, if the user is not in the directory in which these files are found). The images can, of course, be renamed if desired.

C Customizing Your PUNDIT User Environment

Because PUNDIT is written in Quintus Prolog, we can use one of its features to make it easy to customize PUNDIT for individual use. When Prolog first starts up, it checks in the user's home directory for a file named `prolog.ini`. If such a file exists, Prolog will compile it into its current image. Using this feature, we can instruct Prolog to automatically set PUNDIT switches to those settings that we find most convenient. In Figure 10 is an example of one such `prolog.ini`. The example code first checks to see if Prolog is running a PUNDIT image; if it is, switches are set to the desired settings (in this case, to those most convenient for grammar development). Observe in particular that the switch `translated_grammar_in_use` is turned on only if `translated_grammar_present` is already on. At the end, a procedure is called which displays the current switch settings.

```
turn_on_initial_switches:-
    recorded(toggle,switches_are_defined,_),
    !,
    (toggle(translated_grammar_present)->
        turn_on(translated_grammar_in_use);
        true),
    turn_on(parse_tree),
    turn_off(selection),
    ssucceed,
    turn_off(show_isr),
    turn_off(semantics),
    turn_off(text_mode),
    turn_off(summary),
    show_herald.

turn_on_initial_switches.

:- turn_on_initial_switches.
```

Figure 10: Sample `prolog.ini` file

D PUNDIT Files and Dependencies

D.1 Files

Listed below are the core and domain-specific files which comprise the PUNDIT software accompanying this document. By convention, domain-specific files are prefixed with the name of the domain.

- Core Files
  - Lexical
    - **dictisr.pl** - the core lexicon
    - **entries.pl** - the Lexical Entry Procedure
    - **lookup.pl** - lexical lookup
    - **reader.pl** - procedures to read input
    - **readin.pl** - load or update the lexicon
    - **shapes.pl** - shape descriptors
    - **tables.pl** - lexical entry options
  - Syntax
    - Grammar
      - **bnf.pl** - bnf definitions
      - **compile_types.pl** - [created automatically]
      - **compute_types.pl** - compute atomic grammar nodes
      - **conj_restr.pl** - grammar restrictions for conjunction
      - **count_off.pl** - counting procedure
      - **count_on.pl** - counting procedure
      - **counting.pl** - procedures for grinding and counting
      - **interpreter.pl** - grammar interpreter
      - **lspops.pl** - elementary restriction operators
      - **meta.pl** - meta grammar for conjunction
      - **path.pl** - navigate the parse tree
      - **prune.pl** - dynamic pruning of grammar options
      - **restrictions.pl** - restrictions
      - **routines.pl** - basic syntactic routines for grammar
      - **translated_grammar.pl** - [created automatically]
      - **translator.pl** - grammar translator
      - **types.pl** - type definitions for grammar
      - **update.pl** - grammar update procedures

      - **xor.pl** - *exclusive or* mechanism for grammar options
    * Intermediate Syntactic Representation
      - **compute_trans.pl** - compute ISR
      - **isr_lexical.pl** - ISR information for terminal symbols
      - **isr_ops.pl** - ISR operator definitions
      - **semproc.pl** - simplify ISR translation
      - **show_isr.pl** - display procedures for the ISR
    * Selection
      - **selection_dcg.pl** - Selection DCG for analyzing ISR
      - **selection_query.pl** - Selection user interface
      - **selection_restr.pl** - restrictions which call Selection DCG
      - **selection_tools.pl** - Selection tools
      - **selection_top_level.pl** - record and erase parsed sentences
      - **selection_utilities.pl** - Selection utilities
  - Semantics
    * **adjunct_analysis.pl** - analyze sentence adjuncts
    * **filter.pl** - prepare ISR for semantic analysis
    * **np_int.pl** - noun phrase semantics
    * **quantifiers.pl** - *quantifier binding procedures*
    * **semantics.pl** - the Semantic Interpreter
    * **world.pl** - general knowledge base procedures
  - Pragmatics
    * **discourse_rules.pl** - manage discourse and focus information
    * **np_ext.pl** - Reference Resolution
    * **time.pl** - Time Analysis
  - Database Application
    * **entry_generator.pl** - create database relations
  - Utilities
    * **access.pl** - ISR accessor functions
    * **edit.pl** - Prolog Structure Editor
    * **qprolog15.pl** - code specific to Quintus Prolog
    * **rdb_remove.pl** - remove entries from recorded database
    * **show.pl** - display ISR, IDR, db relations, etc.
    * **switches.pl** - manage PUNDIT switches
    * **testing.pl** - software testing utility (not for MUCK
    * **time_display.pl** - temporal relations display procedures

    * **trace_messages.pl** - semantics trace messages
    * **utilities.pl** - general-purpose procedures
    * **vax_menus.pl** - menu facility
    * **vax_show.pl** - top-level non-window display procedures
    * **ws_support.pl** - windowing system procedures
  - Other
    * **compilePundit** - build a PUNDIT image
    * **demo_top_level.pl** -
    * **op_defs.pl** - operator declarations
    * **punt.pl** - on-line PUNDIT help
    * **top_level.pl** - PUNDIT front-end

- Domain-Specific Files for the MUCK Domain
  - Lexical
    * **muck_dictisr.pl** - incremental lexicon
    * **muck_shapes.pl** - shape descriptors
  - Syntax
    * Grammar
      · **compile_types.pl** - [created automatically]
      · **muck_bnf.pl** - updates to the core bnf file
      · **muck_restrictions.pl** - restrictions
      · **translated_grammar.pl** - [created automatically]
    * Selection
      · **muck_selection_db.pl** - selectional patterns
      · **SELECTIONAL_PATTERNS.pl** - [created automatically by Selection]
      · **USER_CORPUS.pl** - [created automatically by Selection]
  - Semantics
    * **muck_rules.pl** - semantics rules
    * **muck_world.pl** - the knowledge base
  - Pragmatics
    * **muck_time.pl** - temporal operators and rules
  - Database Application
    * **muck_entry_generator.pl** - customized version of core file
    * **muck_db_structure.pl** - database definition
    * **muck_db_mapping.pl** - database mapping
  - Summary Application

    * **muck_summary.pl** - create summaries (empty file)
  - Other
    * **compileMuck** - build MUCK image
    * **muck_top_level.pl** - message entry front-end
    * **muck_working.pl** - message corpus

D.2 Dependencies

While most PUNDIT files can be loaded in any order, certain files and classes of files must be loaded in a specific order for PUNDIT to run correctly. These ordering dependencies arise for three main reasons:

1. Compilation of domain-specific files is designed to follow compilation of domain-independent files. For example, certain core procedures may be abolished and redefined in a domain-specific file; if changes are made to the core file and it is recompiled in a domain image, the domain-specific file must be recompiled as well.

2. Some of PUNDIT's data are stored in the Prolog internal database, and multiple compilations of certain files will result in duplicate database entries. The relevant files are: the core and domain-specific versions of the grammar and the lexicon (**bnf.pl** and **dictisr.pl**), and the domain selectional patterns and message corpus.

3. Certain operations in PUNDIT are performed at compile time. These include metarules for the grammar, translating the grammar, and computing the types of nonterminals in the grammar. These operations must be done in order.

If, in the course of development, you wish to compile a new version of the grammar, lexicon, selectional database or message corpus, you must first remove the internal database entries generated by the compilation of the previous version. This can be done most simply by calling the procedure **rdb_remove** (see Section 4), which removes *all* database entries of a specified type.

Compiling changes to selectional patterns: selectional patterns reside in two files: **<domain>_selection_db.pl** and **SELECTIONAL_PATTERNS.pl**. The latter is created automatically in any directory in which you have run a PUNDIT image with the **selection** switch on, while the former resides in the main domain directory, is maintained by hand, and is compiled into the standard domain image. If you wish to retain the selectional patterns which were originally compiled into the image and to add your personal selectional patterns, compile **<domain>_selection_db.pl** and **SELECTIONAL_PATTERNS.pl**, in that order. Otherwise, compile only the relevant file.

Compiling changes to the message corpus: the message corpus is not compiled into either the core PUNDIT image or the domain image; instead, it is automatically compiled

into your image when you first invoke the `punait` command. Therefore, if you have modified this file, you need not recompile it yourself. The system supports personal versions of the corpus: if the file `<domain>_working.pl` exists in the directory in which you are running an image, that is the file which will be compiled. If it does not exist, the file in the main domain directory will be compiled.

Loading changes to the lexicon: multiple lexicon files exist. The core PUNDIT lexicon (`dictisr.pl`) resides in the core PUNDIT directory and is incorporated into the core PUNDIT image; the domain-specific lexicon (`<domain>_dictisr.pl`) resides in the domain directory and is incorporated into the domain image. Since domain images are built from core images, a domain image contains lexical entries from *both* the core lexicon and the domain lexicon, loaded in in that order. In addition, you may have one or more personal lexicon files created by using the Lexical Entry Procedure. By running `rdb_remove` to remove lexical entries, you will have removed *all* lexical entries, regardless of the file in which they originated. You will now need to use the `readIn` procedure, and load the relevant lexicon files in sequence.

Implementing changes to the grammar:

1. Read in new grammar file
2. Meta-Rules—run `gen_conj/0`.
3. Translate the grammar to Prolog—run `translate_grammar/1`, whose argument is a file name (generally `translated_grammar.pl`).
4. Compile the translated grammar—compile the file named above.
5. Compute the types of the grammar nonterminals—compile the file `compute_types.pl`.

These steps must be performed in the order listed, except that step 5 may be performed any time after step 2. Step 2 may be skipped if you do not wish to parse sentences containing conjunction. Skip *both* steps 3 and 4 if you wish to parse with the grammar interpreted (at a significant performance loss). Generally speaking, you will always need to recompile `compute_types.pl`.

Compiling changes to files which do not update the recorded database : certain files exist in core and domain-specific versions (e.g. `shapes.pl` and `<muck>_shapes.pl`). The core versions reside in the core PUNDIT directory and are incorporated into the core PUNDIT image; the domain-specific versions reside in the domain directory and are incorporated into the domain image. Since the domain image is built from the core image, domain-specific files are compiled on top of core files. If you are working in a domain image and have changed a file which exists in both core and domain-specific versions, you will need to recompile both, in that order. Otherwise, simply recompile the relevant file.

E PUNDIT Bibliography

E.1 Background Reading

Dahl, Deborah A. *The Structure and Function of One-Anaphora in English.* PhD thesis, University of Minnesota, 1984; Indiana University Linguistics Club, 1985.

Hirschman, L. Discovering Sublanguage Structures. In Kittredge, R. and Grishman, R. (editors), *Sublanguage: Description and Processing.* Lawrence Erlbaum Assoc., Hillsdale, NJ, 1986.

Palmer, Martha. *Driving Semantics for a Limited Domain.* PhD thesis, University of Edinburgh, 1985.

Palmer, Martha S. *Semantic Processing for Finite Domains.* To appear as a volume in *Studies in Natural Language Processing,* Cambridge University Press, editor, Aravind Joshi, 1988.

Sager, Naomi. *Natural Language Information Processing: A Computer Grammar of English and Its Applications.* Addison-Wesley, 1981.

E.2 Papers and Presentations

Dahl, Deborah A. Focusing and Reference Resolution in PUNDIT. In *Proceedings of the 5th International Conference on Artificial Intelligence.* Philadelphia, PA, August 1986.

Dahl, Deborah A. Determiners, Entities, and Contexts. In *Proceedings of TINLAP-3.* Las Cruces, NM, January 1987.

Dahl, Deborah, Dowding, John, Hirschman, Lynette, Lang, François, Linebarger, Marcia, Palmer, Martha, Passonneau, Rebecca, and Riley, Leslie. *Integrating Syntax, Semantics, and Discourse.* Darpa Natural Language Understanding Program. R&D Status Report, Unisys Defense Systems, May 14, 1987.

Dahl, Deborah A. Integration of Semantics and Pragmatics in the Computational Analysis of Nominalizations. Colloquium presented to the Department of Computer Science, The Pennsylvania State University, October, 1987.

Dahl, Deborah A., Palmer, Martha S., and Passonneau, Rebecca J. Nominalizations in PUNDIT. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics.* Stanford University, Stanford, CA, July 1987.

Dahl, D. Natural Language Processing for Database Generation: The PUNDIT System. Paper presented at AI West, May, 1988, Long Beach California.

Dowding, John and Hirschman, Lynette. Dynamic Translation for Rule Pruning in Restriction Grammar. In *Proceedings of the 2nd International Workshop on Natural Language Understanding and Logic Programming.* Vancouver, B.C., Canada, 1987.

Grishman, R. and Hirschman, L. PROTEUS and PUNDIT: Research in Text Understanding. *Computational Linguistics* 12(2):141-45, 1986.

Hirschman, Lynette. Conjunction in Meta-Restriction Grammar. *Journal of Logic Programming* 4:299-328, 1986.

Hirschman, Lynette. Natural Language Interfaces for Large Scale Information Processing. Technical Advisory Panel Meeting for the Transportation Systems Center, Department of Transportation. Boston, MA, May, 1987.

Hirschman, Lynette, Tutorial on Natural Language and Logic Programming. 1987 Logic Programming Symposium, San Francisco, Aug. 31-Sept. 4, 1987.

Hirschman, L. A Meta-Treatment of wh-Constructions. To be presented at META 88 Workshop on Meta Programming in Logic Programming.

Hirschman, Lynette, Dahl, Deborah, Dowding, John, Lang, François-Michel, Linebarger, Marcia, Palmer, Martha, Riley, Leslie, and Schiffman, [Passonneau] Rebecca. The PUNDIT Natural Language Processing System. Presented at the Eleventh Annual Penn Linguistics Colloquium, Philadelphia, PA, February, 1987.

Hirschman, L., Hopkins, W.C., Smith, R.C. Or-Parallel Speed-up in Natural Language Processing: A Case Study. To be presented at the 5th International Logic Programming Conference, Seattle, August, 1988.

Hirschman, L. and Puder, K. Restriction Grammar in Prolog. In *Proceedings of the First International Logic Programming Conference,* pages 85-90.

Hirschman, L. and Puder, K. Restriction Grammar: A Prolog Implementation. In Warren, D.H.D. and Van Caneghem, M. (editors), *Logic Programming and its Applications,* pages 244-261. Ablex Publishing Corp., Norwood, NJ, 1986.

Lang, François-Michel and Hirschman, Lynette. Improved Portability and Parsing through Interactive Acquisition of Semantic Information. In *Proceedings of the Second Conference on Applied Natural Language Processing.* Austin, TX, February 1988.

Linebarger, Marcia C., Dahl, Deborah A., Hirschman, Lynette, and Passonneau, Rebecca J. Sentence Fragments Regular Structures. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics.* Buffalo, NY, June 1988.

Palmer, Martha S., Dahl, Deborah A., Passonneau, Rebecca J., Hirschman, Lynette, Linebarger, Marcia, and Dowding, John. Recovering Implicit Information. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics.* Columbia University, New York, August 1986.

Palmer, Martha, and Linebarger, Marcia. Status of Verb Representations in PUNDIT. Presented at Theoretical And Computational Issues in Lexical Semantics, Brandeis University, Waltham, Mass, April 21-24, 1988.

Palmer, Martha, Hirschman, Lynette, and Dahl, Deborah. Text Processing Systems. June 1988. Tutorial presented at the 26th Annual Meeting of the Association for Computational Linguistics, Buffalo New York.

Passonneau, Rebecca J. Situations and Intervals. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, pages 16-24. 1987.

Passonneau, Rebecca J. A Computational Model of the Semantics of Tense and Aspect. *Computational Linguistics* (forthcoming), 1988.

E.3 Technical Documentation

Ball, Catherine N., Dahl, Deborah A., Dowding, John, Hirschman, Lynette, Linebarger, Marcia, Palmer, Martha, and Passonneau, Rebecca. *PUNDIT Tutorial Notes.* Internal document, Unisys Corporation, 1987.

Lang, François-Michel. *A User's Guide to the Selection Module.* LBS Technical Memo 68, Unisys Corporation, 1987.

Linebarger, Marcia C. *A Guide to Object Options in PUNDIT.* Technical Report, Unisys Corporation, 1988.

Riley, Leslie. *A Guide to the PUNDIT Lexical Entry Procedure.* Technical Report, Unisys Corporation, 1988.

Riley, Leslie and Dowding, John. *The Prolog Structure Editor.* LBS Technical Memo 29, Unisys Corporation, 1986.

Schiffman (Passonneau), Rebecca J. *Designing Lexical Entries for a Limited Domain.* LBS Technical Memo 42, Unisys Corporation, April 1986.

References

[Dahl 84] Dahl, Deborah A. *The Structure and Function of One-Anaphora in English.* PhD thesis, University of Minnesota, 1984.

[Lang 87] Lang, François-Michel. *A User's Guide to the Selection Module.* LBS Technical Memo 68, Unisys Corporation, 1987.

[Linebarger 88] Linebarger, Marcia C. *A Guide to Object Options in PUNDIT.* Technical Report, Unisys Corporation, 1988.

[Passonneau 87] Passonneau, Rebecca J. Situations and Intervals. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, pages 16–24. 1987.

[Riley 86] Riley, Leslie and Dowding, John. *The Prolog Structure Editor.* LBS Technical Memo 29, Unisys Corporation, 1986.

[Riley 88] Riley, Leslie. *A Guide to the PUNDIT Lexical Entry Procedure.* Technical Report, Unisys Corporation, 1988.

[Sager 81] Sager, Naomi. *Natural Language Information Processing: A Computer Grammar of English and Its Applications.* Addison-Wesley, 1981.

PUNDIT
Lexical Entry Procedure
User's Guide*

Version 1.0

August 10, 1988

Unisys Logic-Based Systems
Paoli Research Center
P.O. Box 517, Paoli, PA 19301

*This work has been supported by DARPA contract N00014-85-C-0012, administered by the Office of Naval Research.

Contents

1 Introduction

The Lexical Entry Procedure has been designed to provide consistency, completeness, and speed of entry for new words. The procedure elicits relevant linguistic information from the user, computes dependencies between attributes, and prompts for morphologically related forms (offering a "guess" as to the correct form). The program then automatically creates a set of related dictionary entries, with as much structure-sharing among the entries as possible. Before the entries are actually entered in the database or written to a file, the user may inspect and edit any entries created.

2 Getting Started

The Lexical Entry Procedure may be invoked automatically or explicitly.

When PUNDIT fails to find a definition for some token in the input stream, and the switch **enter_new_word**[1] is on, lexical lookup will trap to the following menu:

```
1. Respell word
2. Add dictionary entry
3. Word is a proper noun
4. Quit
```

Option 2 invokes the Lexical Entry Procedure. The procedure may also be called directly by executing the goal:

```
?- recordNewEntry(<word>).
```

where **<word>** is the word whose lexical entry you wish to create. Note that the Lexical Entry Procedure is designed only for the entry of new words: if you wish to revise the lexical entries for an existing word, you must use the procedure **edit_word** (see [Riley 86]) instead.

Upon entering the Lexical Entry Procedure, the user is given the option to save the resulting lexical entries into a file in the current working directory. The name of this file is automatically generated as **<domain>_dictisr.pl.<id>**[2]. Entries in this file must then be added to the PUNDIT lexicon.

The user is next prompted for the root of the word, which is the most basic form of the word in the same grammatical category. Here, it is best to think in terms of inflectional rather than derivational morphology: for example, the root of *thought* in *They thought about it* is *think*, while the root of *thought* in *That was a good thought* is *thought*.

After entering the root and (optionally) any abbreviations, the user is prompted for morphological and grammatical information.

---

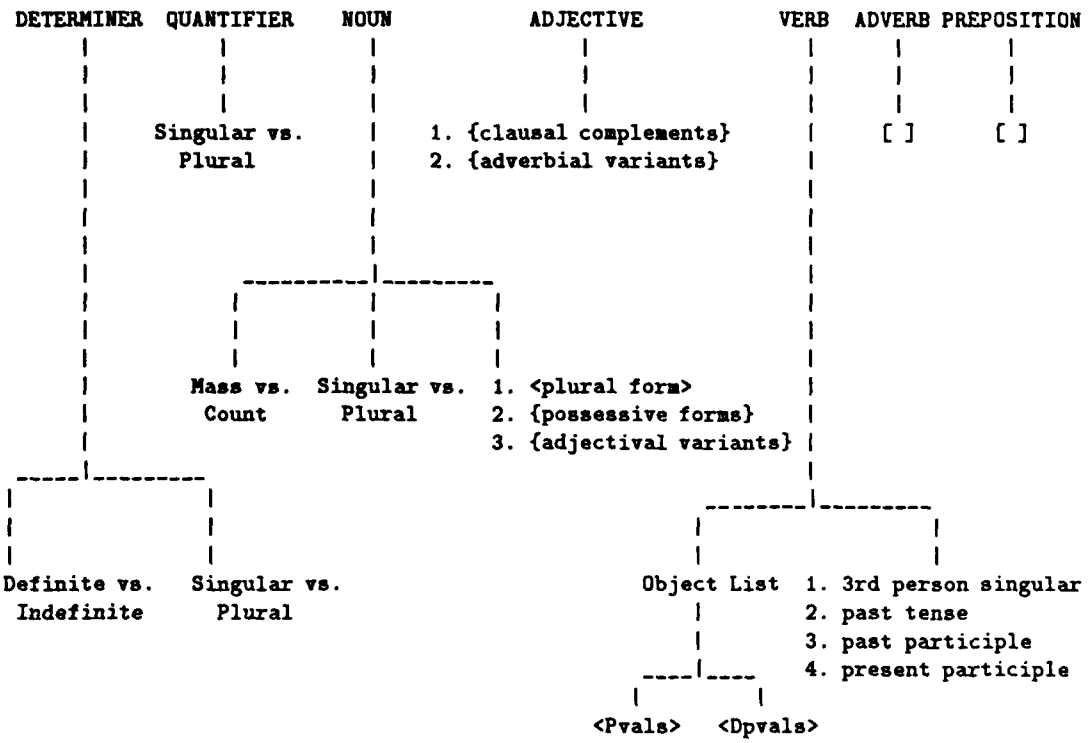
[1] For more information on this and other PUNDIT switches, please consult [Ball 88].

[2] For example, a file created using the Lexical Entry Procedure in the MUCK domain on August 10 might be named **muck_dictisr.pl.10Aug1313**. The last element of the name distinguishes multiple files created on the same date.

The highest-level menu[3] in the Lexical Entry Procedure is the menu of grammatical categories, or word classes:

```
Word Classes
 1. Determiner          2. Quantifier          3. Adjective
 4. Noun                5. Verb                6. Adverb
 7. Preposition         8. More/Other
```

The next sections cover major word classes and their features in detail. The diagram below shows the features and morphological information which are collected for each word class. Items enclosed in { } are optional, while items enclosed in < > reflect information that the user may or may not be asked to provide, depending on previous choices.



[3] A few notes on using menus:

- When prompted for a list of items, the user may enter either a single item or a Prolog list, followed by a period. The examples on the following pages illustrate both possibilities.
- The appearance of `More/Other` in a menu indicates that the choices offered may not be exhaustive. By selecting this option, the user may input items of his or her choice.

3 Major Word Classes

3.1 Determiners

Determiners are classified according to definiteness and number. A sample definition for *another*:

```
*** another - has been classified as a determiner
              A definite or indefinite determiner -- use menu

 1. Definite
 2. Indefinite
 3. Neither

Please choose an item -- 2.

              Singular or plural -- use menu

 1. Singular
 2. Plural
 3. Neither

Please choose an item -- 1.
```

The following lexical entry is created:

```
:(another,root:another,[t:[indef,singular]])
```

Each lexical entry consists of the citation form, followed by the root, followed by a list of lexical classes and their attributes. The letter *t* in this lexical entry designates the class of determiners[4].

3.2 Quantifiers

Quantifiers are classified according to number. A sample definition for *some* is given below:

```
*** some - has been classified as a quantifier

              Singular or Plural -- use menu

 1. Singular
 2. Plural
 3. Neither

Please choose an item -- 2.
```

[4]The reader may find it useful to consult [Fitzpatrick 81] for a more detailed discussion of this and other word classes in the context of a related system.

The following lexical entry is created:

```
:(some,root:some,[q:[plural]]):
```

3.3 Adjectives

The Lexical Entry Procedure asks two questions about adjectives:

1. **Can this adjective take a clausal complement (y or n)?**
2. **Does <*ADJECTIVE*> have a(n) adverb form (y or n)?**

If the answer to the first question is **y**, the user will be asked to classify the valid clausal complements as one or more of the following:

- **asent1**
  Takes a tensed clause complement. Subject must be pleonastic *it*.
  Example: *It is clear that he is tired.*
- **asent3**
  Takes a tensed clause complement. Subject need not be pleonastic *it*.
  Example: *I am glad that she won.*
- **aasp:[equi_adj]**
  Takes an infinitival complement with *equi* argument structure.
  Example: *They are eager to please.*
- **aasp:[raising_adj]**
  Takes an infinitival complement with *raising* argument structure (see the attached guide to PUNDIT object types for the distinction between equi and raising).
  Example: *She is certain to be re-elected.*

If the user answers **y** to the second question, the procedure will prompt for the adverbial form.

A sample definition for the adjective *certain* is given below:

```
*** certain - has been classified as an adjective.
Can this adjective take a clausal complement (y or n)? y

Choose whatever features apply:
 1. asent1.            Eg: It is likely that he will repair it.
 2. asent3.            Eg: He is glad that it is repaired.
 3. aasp:[equi_adj].   Eg: He is unable to repair it.
 4. aasp:[raising_adj]. Eg: He is likely to repair it.

Please choose a list of items -- [1,2,4].

Does certain have a(n) adverb form (y or n)? y

Enter the adverb of certain -- certainly.
Is certainly correct? y
```

The following lexical entries are created:

```
:(certain,root:certain,[adj:[asent1,asent3,aasp:[raising_adj]]])
:(certainly,root:certain,[d])
```

3.4 Nouns

A noun is first classified as mass or count. If the noun is a count noun, the procedure prompts for number information and plural form (it is assumed that the root is singular). If it is mass noun, the procedure asks whether it can have a plural form different from its singular form. The user is then asked about adjectival and possessive forms. Sample definitions for the count noun *woman* and the mass noun *mud* are given below.

```
*** woman - has been classified as a noun.
              Mass or Count Noun -- use menu

 1. Count Noun
 2. Mass Noun
 3. Other

Please choose an item -- 1.

              Singular or Plural -- use menu

 1. Singular
 2. Plural
 3. Neither

Please choose an item -- 1.

Is "womans" the plural form of woman? n

Enter the correct form -- women.
Is women correct? y

Does woman have a(n) adjective form (y or n)? n

Does woman have a singular possessive form (y or n)? y

Is "woman's" the singular possesssive form of woman? y

Does woman have a plural possessive form (y or n)? y

Is "womans'" the plural possesssive form of woman? n

Enter the correct form -- 'women''s'.
```

```
Is women's correct? y
```

Note in particular the manner of entering forms with apostrophes (*woman's*). Because Prolog is being used to read the user's response, an apostrophe must be entered as two single quotes, and the entire word must be enclosed in single quotes: `'woman''s'` instead of `woman's`.

The following lexical entries are created:

```
:(woman,root:woman,[n:[11,singular],11:[ncount1]])
:(women,root:woman,[n:[11,plural]])
:(woman's,root:woman,[ns:[11,singular]])
:(women's,root:woman,[ns:[11,plural]])
```

A sample definition for the mass noun *mud*:

```
*** mud - has been classified as a noun.
              Mass or Count Noun -- use menu

 1. Count Noun
 2. Mass Noun
 3. Other

Please choose an item -- 2.


Can this mass noun have a plural form different from its singular form (y or n)? n

Does mud have a(n) adjective form (y or n)? y

Enter the adjective of mud -- muddy.
Is muddy correct? y

Does mud have a singular possessive form (y or n)? y

Is "mud's" the singular possesssive form of mud? y

Does mud have a plural possessive form (y or n)? n
```

The following lexical entries are created:

```
:(mud,root:mud,[n:[11],11:[collective]])
:(muddy,root:mud,[adj])
:(mud's,root:mud,[ns:[11,singular]])
```

3.5 Verbs

When entering a verb, the user is first prompted for the object types. For a complete description of the object types currently implemented, see [Linebarger 88], attached as an appendix to this guide.

If the object contains a preposition or particle, the user will be prompted to enter valid preposition values (**pvals**) and particle values (**dpvals**). Note that the passive objects are automatically computed, as well as the **pvals** and **dpvals** for the passive objects.

The user is also asked to enter certain morphological variants of the verb, such as the past tense and participial forms. A sample definition for the verb *think* is shown on the following page.

```
*** think - has been classified as a verb form

              Choose the objectlist -- use menu

 1. nullobj     2. nstgo       3. pn          4. npn         5. pnn
 6. objbe       7. eqtovo      8. tovo        9. ntovo      10. objtovo
11. thats      12. assertion  13. pnthats    14. svo        15. c1should
16. pnthatsvo  17. snwh       18. nswh       19. nthats     20. sven
21. nn         22. sobjbe     23. na         24. astg       25. dstg
26. dp1        27. dp2        28. dp3        29. dp1pn      30. dp2pn
31. dp3pn      32. dpsn       33. More/Other
Please choose a list of items -- [1,3,11,27,28,12,22].

     Classify the prepositions for the PN object of "think" --

 1. about       2. after       3. against     4. at          5. by
 6. from        7. for         8. in          9. into       10. of
11. off        12. on         13. over       14. to         15. up
16. upon       17. with       18. More/Other
Please choose a list of items -- 1.

     Classify the particles for the DP2 object of "think" --

 1. about           2. around          3. away           4. by
 5. down            6. in              7. off            8. on
 9. out            10. over           11. through       12. to
13. up             14. More/Other
Please choose a list of items -- 13.

     Classify the particles for the DP3 object of "think" --

 1. about           2. around          3. away           4. by
 5. down            6. in              7. off            8. on
 9. out            10. over           11. through       12. to
13. up             14. More/Other
Please choose a list of items -- 13.
```

```
Is "thi :s" the present third person singular form of think? y

Is "thinked" the past tense of think? n

Enter the correct form -- thought.
Is thought correct? y

Is "thought" the past participle of think? y

Is "thinking" the present participle of think? y
```

The following lexical entries are created:

```
:(think,root:think,[v:[12],tv:[12,plural],
        12:[objlist:[thats,nullobj,assertion,sobjbe,pn:[pval:[about]],
                     dp2:[dpval:[up]],dp3:[dpval:[up]]]]]).

:(thinks,root:think,[tv:[12,singular]]).

:(thought,root:think,[tv:[12,past],ven:[14],
          14:[12,pobjlist:[assertion,objbe,thats,dp1:[dpval:[up]],
                           p:[pval:[about]]]]]).

:(thinking,root:think,[ving:[12]]).
```

4 Exiting the Lexical Entry Procedure

After all the entries have been created, the user is given the opportunity to inspect each entry and to do one of the following:

```
1. Enter It
2. Do Not Enter It
3. Edit It
```

Option 1 will cause the new lexical entry to be entered in the Prolog database and written to a file (if the user has so directed the procedure). Choosing option 2 will cause the entry to be ignored. If option 3 is chosen, the Prolog Structure Editor will be invoked on that lexical entry (see [Riley 86] for more details). Note that no action is taken until one of these choices is made for each entry.

If you have chosen to write the lexical entries to a file, you may now wish to add these entries to the core lexicon or the domain lexicon. This must be done manually. You may then wish to load the entire lexicon into an image for testing; consult [Ball 88] for the procedures to be followed.

References

[Ball 88] Ball, Catherine N., Dowding, John, Lang, François-Michel, and Weir, Carl. *PUNDIT User's Guide.* Technical Report, Unisys Corporation, 1988.

[Fitzpatrick 81] Fitzpatrick, Eileen and Sager, Naomi. Appendix 3: The lexical subclasses of the LSP String Grammar. In Sager, Naomi (editor), *Natural Language Information Processing: A Computer Grammar of English and Its Applications*, pages 322–374. Addison-Wesley, Reading, Mass., 1981.

[Linebarger 88] Linebarger, Marcia C. *A Guide to Object Options in PUNDIT.* Technical Report, Unisys Corporation, 1988.

[Riley 86] Riley, Leslie and Dowding, John. *The Prolog Structure Editor.* LBS Technical Memo 29, Unisys Corporation, 1986.

A Object Options

A Guide to Object Options in PUNDIT*

Marcia Linebarger

August 10, 1988

*This work has been supported by DARPA contract N00014-85-C-0012, administered by the Office of Naval Research.

Contents

1 Introduction

This document describes the current object options of the grammar, with the corresponding passobj (passive object) options and ISRs (Intermediate Syntactic Representations – see below), and with some very limited annotations on their structural quirks, semantics, raison d'être, and so forth. The numbering of object options below is the same as that in the Lexical Entry Procedure, and these notes are intended for use during entry of new lexical items. Object options which are restricted to one or two verbs (such as BE_AUX, VENO, and VO, associated with the auxiliaries *be*, *have*, and modals) are not included in this list, because we assume that most verbs with these subcategorizations have already been entered in the lexicon. Such object types may be assigned to a new verb by choosing `Other` in the Lexical Entry Procedure menu.

1.1 Handling of Passive in the Lexicon

The parse tree built by PUNDIT represents surface structure; transformations such as passivization and wh-movement are not 'undone' at this level. Thus verbs must be subcategorized for the objects they take in both active and passive. (Note on terminology: objects of the verb in its active form are called **`object`**; the list of a verb's objects in the lexicon is called the **`objlist`**. Similarly, passive objects are called **`passobj`**, and the list of a verb's passive objects in the lexicon is called the **`pobjlist`**. Note the systematic ambiguity of the word 'object'.) Because the correlation between an active and a passive object is predictable, the Lexical Entry Procedure automatically computes the passobj on the basis of the active objects selected. Verbs which do not passivize receive no pobjlist whatsoever in the lexicon; they should not be subcategorized for NULLOBJ in the passive. The *by*-phrase, if present, is parsed as a sentence adjunct rather than a passobj. Note that although some active object options (e.g., NULLOBJ) are never associated with a corresponding passive object, since they never passivize, others may or may not be; since the Lexical Entry Procedure automatically computes the corresponding passobj for any object type which passivizes, it is up to the user to edit out of the lexical entry any unacceptable passobj.

1.2 The ISR

Although the parse tree represents surface structure, the ISR is a somewhat more abstract level of syntactic representation, which, like the 'deep structure' of transformational grammar, provides a more transparent representation of argument structure. For example, the surface subject of the passive is represented in the ISR as the object of the verb. As in many current syntactic theories, the subject position of a passive ISR remains unfilled (in PUNDIT, it is filled with the dummy element **`passive`**), and it is the function of semantic rules to determine whether an element in a *by*-phrase may fill the semantic role which would be assigned to the subject. Thus, at least for the object, active and passive sentences can be interpreted by the same semantic mapping rules. In some cases, the ISRs of passive sentences diverge significantly from the surface structure in order to bring about this parallelism between active and passive; for example, the ISR for a pseudopassive such as *The patient was operated on* reconstructs the prepositional phrase. Thus the surface parse tree provides the bare preposition *on* as object of the verb, while the ISR provides the prepositional phrase *on the patient* as object.

The ISR also fleshes out the argument structure of constructions such as equi and raising, as seen in connection with object types EQTOVO, TOVO, and OBJTOVO below; and it regularizes the surface

order of object types which differ from one another only in the order of their components (such as NPN and PNN, or DP2 and DP3).

Because there are such divergences between the ISR and the surface parse, and because the ISR plays an important role as the interface between syntax and semantics, the ISRs associated with each object type and its passivized counterpart are given below. For ease of exposition, only the prettyprinted ISR is displayed.

1.3 On pvals and dpvals

Object types containing prepositions can be subcategorized for particular prepositions, via **pval** sublists in the lexicon; object types containing particles can be subcategorized for specific particles via **dpval** lists in the lexicon. The Lexical Entry Procedure queries the user to create these lists where appropriate.

2 Object Options

2.1 NULLOBJ

A verb which takes no complement at all is subcategorized for NULLOBJ. Example: *The pump failed*, which receives the following ISR:

```
OPS:   past
VERB:  fail
SUBJ:  the pump (sing)
```

Such verbs do not passivize, hence there is no corresponding passobj.

2.2 NSTGO

This is the simple transitive verb option, a noun phrase non-predicative direct object. Example: *She repaired the sac*, which receives the following ISR. The direct object receives the semlabel **obj**. (Semlabels are applied to elements in the ISR to label those grammatical functions which play a role in semantic rules. In the prettyprinted ISRs, the semlabels of all postverbal elements appear in capital letters, e.g. **SUBJ**: in the example below.)

```
OPS:   past
VERB:  repair
SUBJ:  pro: she (sing)
OBJ:   the sac (sing)
```

The passobj counterpart of NSTGO is NULLOBJ, as in *The sac was repaired (by her).* The *by*-phrase is parsed as a sentence adjunct; this is not evident in the ISR below because the ISR (for reasons having to do with the functioning of the semantic interpreter) fails to indicate whether a prepositional phrase occurs as a sentence adjunct or a verb object.

```
OPS:   past
VERB:  repair
SUBJ:  passive
OBJ:   the sac (sing)
PP:    by
       pro: her (sing)
```

Note that the surface subject is represented as the object in the ISR. The subject position of the ISR is filled with the dummy element **passive**.

2.3 PN

This is a prepositional phrase object. Example: *They operated on him*:

```
OPS:   past
VERB:  operate
SUBJ:  pro: they (pl)
PP:    on
       pro: him (sing)
```

Corresponding passobj: isolated preposition. Example: *He was operated on*; in the ISR, the prepositional phrase is reconstructed:

```
OPS:   past
VERB:  operate
SUBJ:  passive
PP:    on
       pro:   he (sing)
```

When do we want PN to be analysed as an object option rather than a sentence adjunct (SA)? As far as I can tell, the following are the most relevant cases in which the PN object is subcategorized for in this system:

(a) The verb is unacceptable with NULLOBJ, and PN will suffice. E.g., **He told* (ignoring elliptical reading). But *He told of great adventures.*

(b) The VERB + PN has an idiomatic meaning (or just feels like a unit): *the surgeon operates on the patient* and *the surgeon operates on the table* represent, under their most plausible readings, the PN object and SA attachments respectively. Similarly: *Bill turned into the side street* (SA expressing where he turned) vs. *Bill turned into an orangutang* (PN object).

The possibility of a pseudopassive doesn't seem to be a motivating factor: *sleep* in our lexicon isn't subcategorized for *in* or *on*, etc., yet you can say *That bed was slept in by George Washington* or *This floor has been slept on by countless fatigued partygoers.* If a verb with PN object can passivize at all, as above, its passobj will be a P (at the moment this passobj is not listed under very many verbs in the lexicon.) Thus it is currently an unsolved problem how to treat pseudopassives

corresponding to active sentences in which the PN is in SA as in the *sleep* example above: we don't really want to allow P as an SA option generally. So another possibility would be to allow PN object (with no subcategorization for specific lexical items) more freely, automatically generating the PN object possibility for ANY verb which allows pseudopassive. The cost of this is that we lose the way of structurally representing differences such as that between, e.g., *operate on the table* and *operate on the patient.*

2.4 NPN

and

2.5 PNN

NPN consists of an NSTGO followed by a PN, as in *They returned the disk drive to the factory*:

```
OPS:   past
VERB:  return
SUBJ:  pro:   they (pl)
OBJ:   the disk^drive (sing)
PP:    to
       the factory (sing)
```

See above for discussion of when to include the PN in object rather than SA. Another criterion: is there a corresponding PNN object? PNN is the BNF node associated with NPN which has undergone a shifting of the NP, constrained by various stylistic factors such as heaviness. It's one of the unpleasant facets of the grammar we use that this extraposition gets expressed as a different BNF node. Subcategorization for PNN follows redundantly from subcategorization for NPN, since the acceptability of PNN depends not on the verb but on the NP itself. (Compare *He presented to us an enormous chocolate cake iced with yellow daffodils* vs. the much less pleasing *He presented to us a cake.*)

Note that a sequence of NP + PN need not be parsed as NPN; for example, *I found Louise in a state of euphoria* should probably be classed as a SOBJBE (see below), given related sentences such as *I found Louise euphoric, I found Louise a changed woman.* The PN here is predicated of Louise rather than simply being an argument of *find.* In contrast, the PN in *I found Louise on the fourth try* seems more like an SA describing the circumstances of the event of finding Louise, certainly not a predication stating that Louise was *on the fourth try.*

The passobj counterpart of NPN/PNN is PN, as in *The disk drive was returned to the factory*:

```
OPS:   past
VERB:  return
SUBJ:  passive
OBJ:   the disk^drive (sing)
PP:    to
       the factory (sing)
```

(Compare **The factory was returned the disk drive to*: no pseudopassive is possible here except with idiomatic expressions such as *He was given a talking to.*)

2.6 OBJBE

OBJBE, the object type associated with *be* as a main verb, is subcategorized for by verbs other than *be*. OBJBE expands to an NP, an adjective phrase, or a PP; not every verb allows all these expansions, as indicated by bvals in the lexicon. (The Lexical Entry Procedure does not currently solicit bvals.) Examples: *The pump appears inoperative*:

```
OPS:   present
VERB:  appear
SUBJ:  the pump (sing)
ADJ:   inoperative
```

and *She became a field engineer*:

```
OPS:   past
VERB:  became
SUBJ:  pro:  she (sing)
PREDN: a field^engineer (sing)
```

These verbs don't passivize at all, so they have no passobj counterpart (and hence no pobjlist is created for them by the Lexical Entry Procedure.)

Thus an NP following the verb can be analysed either as an NSTGO (*He photographed the President's advisor*) or as an OBJBE (*He became the President's advisor*). This enforces the well-known fact that predicative verbs do not passivize: *The best cars are made by the Japanese* (active form: nstgo) vs. **The best cooks are made by Italians* (active form: objbe).

2.7 EQTOVO

An example of EQTOVO is *The fe wants to repair the disk drive*. EQTOVO corresponds to what is traditionally known as an infinitival complement with subject controlled equi; the subject of the matrix verb is understood to be also the subject of the infinitive. This is made explicit in the ISR, where the matrix subject is copied into the infinitive; the ID variables for the two NPS are identical (a fact which is obscured below because the ISR prettyprinter does not display variables):

```
OPS:   present
VERB:  want
SUBJ:  the field^engineer (sing)
OBJ:   OPS:   untensed
       VERB:  repair
       SUBJ:  the field^engineer (sing)
       OBJ:   the disk^drive (sing)
```

There is no passobj, as these structures do not passivize.

2.8 TOVO

An example of TOVO is *The pump seems to be failing.* The TOVO object corresponds to what is traditionally known as raising; the matrix subject is analysed as an argument of the infinitive, but not of the matrix verb, which has the infinitive as its sole argument. This is made explicit in the ISR, where the reconstructed infinitival clause is the subject:

```
OPS:   present
VERB:  seem
SUBJ:  OPS:   untensed,prog
       VERB:  fail
       SUBJ:  the pump (sing)
```

As for passobj, raising verbs don't passivize, so there is no pobjlist.

As noted above, these two object types EQTOVO and TOVO differ in their argument structure, and hence in their selection properties, differences which are made explicit in the ISR. In the EQTOVO (equi) case, the phonologically null subject of the infinitive undergoes selection with the matrix verb as well as with the verb in the infinitive. That is, *the fe* is really the subject of both *want* and *repair* in *The fe wants to repair the disk drive.* One can run afoul of selection restrictions between this noun and *either* verb: *The number 12 wants _ to be divisible by 3*, and *The cat wants _ to be divisible by 3* are both anomalous, due to violations of selection between the matrix subject and the matrix and embedded predicates, respectively.

For the bare TOVO case, the matrix subject is semantically just the subject of the lower verb; that is, the matrix verb is really a one-place predicate with a clause as its argument. (Thus the ISR subject of *The pump seems to be failing* is not *the pump* but *the pump to be failing.*) There's no selection between the surface NP subject (*the pump*) and this matrix verb (*seem*): whatever can be subject of the infinitival verb V can also be subject of *seem to V...D.* Sager refers to these as aspectual verbs. They include: *seem*, *appear*, *start*, *tend*, *continue*, *come* (as in *It came to rotate*, NOT as in *I come to bury Caesar, not to praise him.* The latter is a purposive TOVO in SA.)

To summarize: with EQTOVO, the matrix subject is an argument of the matrix verb and also of the verb in the infinitive; with TOVO, the matrix subject is an argument only of the lower (infinitival) verb. (The two types correspond to equi and raising, respectively.)

In Sager's grammer, these two categories are conflated. Some existing lexical entries therefore require updating, since this distinction was introduced after PUNDIT's lexicon was established.

2.9 NTOVO

Like OBJTOVO (see below), NTOVO is associated with surface sequences of the form 'NP to VP' following the matrix verb; it corresponds to what is sometimes called 'exceptional case marking (ECM)'. An example of NTOVO is *The factory expects the fe to repair the sac*:

```
OPS:   present
VERB:  expect
SUBJ:  the factory (sing)
OBJ:   OPS:   untensed
```

```
VERB:  repair
SUBJ:  the field^engineer (sing)
OBJ:   the sac (sing)
```

Thus *the field engineeer* is the subject of the clause but is not a direct object of the matrix verb; the factory does not expect the fe, but rather it expects the proposition expressed by the infinitive. (A consequence of this is that pleonastic elements such as *there* may occur in subject position of NTOVO: *I expect there to be unlimited champagne.*)

The passobj counterpart of NTOVO is TOVO, as in *The fe is expected to repair the sac*; the ISR rule associated with TOVO will automatically reconstruct the infinitive *the fe to repair the sac*:

```
OPS:   present
VERB:  expect
SUBJ:  passive
OBJ:   OPS:   untensed
       VERB:  repair
       SUBJ:  the field^engineer (sing)
       OBJ:   the sac (sing)
```

2.10 OBJTOVO

OBJTOVO corresponds to object controlled equi; in *The factory told the fe to repair the pump, the fe* is an argument (indirect object?) of the matrix verb and subject of the infinitive:

```
OPS:   past
VERB:  tell
SUBJ:  the factory (sing)
D_OBJ: the field^engineer (sing)
OBJ:   OPS:   untensed
       VERB:  repair
       SUBJ:  the field^engineer (sing)
       OBJ:   the pump (sing)
```

The semlabel d_obj (dative object, formerly known as inner_obj) is used here to capture the parallelism with *The factory told the fe the truth.*

The passobj counterpart is EQTOVO. The ISR rules associated with EQTOVO reconstruct infinitive as above for *The fe was told to repair the sac*:

```
OPS:   past
VERB:  tell
SUBJ:  passive
D_OBJ: the field^engineer (sing)
OBJ:   OPS:   untensed
       VERB:  repair
       SUBJ:  the field^engineer (sing)
       OBJ:   the sac (sing)
```

Major differences between NTOVO, OBJTOVO: in NTOVO, the subject of the infinitive is an argument ONLY of the lower verb. The entire infinitival clause is itself the argument of the matrix verb. There are no selection restrictions between, e.g., *believe* and *the table* in *I believed the table to be quite attractive.* In OBJTOVO, on the other hand, the noun phrase between the matrix verb and the infinitive is an argument of BOTH matrix and embedded predicates, as demonstrated by the anomaly of *I persuaded the table to seat 6* (violates selectional constraints on *persuade*) and *I persuaded the man to be divisible by 2* (violates selectional constraints on *divisible*). also, NTOVO but not OBJTOVO allows *there* as subject: *I expect there to be a diplomat at the party* (**I persuaded there to be a diplomat at the party*).

PUNDIT does not currently handle the rare cases of subject-controlled equi in verb complements of the form 'NP to VP', as in *Mary promised Louise to arrive on time.* This form of control is largely restricted to the single verb *promise.*

2.11 THATS

and

2.12 ASSERTION

THATS and ASSERTION are both tensed clauses, with and without the complementizer *that*, as in *The fe said that the disk drive was inoperative*:

```
OPS:   past
VERB:  say
SUBJ:  the field^engineer (sing)
OBJ:   OPS:   past
       VERB:  be
       SUBJ:  the disk^drive (sing)
       ADJ:   inoperative
```

Verbs subcategorized for THATS and ASSERTION are automatically subcategorized for these same objects in the passive, given the possibility of pleonastic subjects, as in *It is said that whales are highly intelligent.* Work remains to be done to constrain these cases in the grammar. General note on passobjs with verbs taking clausal objects (ASSERTION, THATS, PNTHATS, SVO, C1SHOULD, SNWH, NSNWH, NTHATS): in Sager, passives with *it* subject (*It was reported that the disk failed*) are not treated as having a clausal passobj. Rather, the clause goes into **rv** at the string level. However, it seems to me that these verbs should all be subcategorized for clausal passobj.

2.13 PNTHATS

This is a PN followed by THATS, as in *The fe reported to the factory that the sac had failed*:

```
OPS:   past
VERB:  report
SUBJ:  the field^engineer (sing)
```

```
PP:    to
       the factory (sing)
OBJ:   OPS:   past,perf
       VERB:  fail
       SUBJ:  the sac (sing)
```

These objects are further subcategorized for pvals, like all PN-containing objects. Not every VERB + PP + CLAUSE structure involves a PNTHATS; for example, *this proves with some certainty that the world is round* should be analyzed as a THATS with preceding PN in SA, while *this proved to everyone that the theory was wrong* should be treated as PNTHATS with PN in OBJECT.

The passobj counterparts are PN and PNTHATS, as in *It was revealed to us yesterday that the company had gone bankrupt* (PNTHATS as passobj), or *That Smith was the culprit was announced to the entire assembly* (PN as passobj).

2.14 SVO

SVO is a tenseless clause; it differs from C1SHOULD (see below) in that (1) SVO never has the complementizer *that*, (2) a pronoun subject of SVO is accusative. Example: *She saw them replace the pump*:

```
OPS:   past
VERB:  saw
SUBJ:  pro:   she (sing)
OBJ:   OPS:   untensed
       VERB:  replace
       SUBJ:  pro:   them (pl)
       OBJ:   the pump (sing)
```

Passivization is not acceptable out of SVO, cf. **They were seen replace the pump.*

2.15 C1SHOULD

This consists of the complementizer *that* followed by SVO, as in *He suggested that it be replaced*:

```
OPS:   past
VERB:  suggest
SUBJ:  pro:   he (sing)
OBJ:   OPS:   untensed
       VERB:  replace
       SUBJ:  passive
       OBJ:   pro:   it (sing)
```

Passobj counterparts: C1SHOULD, as in *It was suggested that we leave early*; and probably NULLOBJ. (My intuitions are unclear on NULLOBJ as passobj here.)

A pronoun subject of C1SHOULD is nominative. The current BNF rule for C1SHOULD requires *that*, but should be generalized to account for *I suggest we leave.*

2.16 PNTHATSVO

This consists of PN followed by C1SHOULD, as in *I suggested to Bill that he write up his investigations.* Pvals are elicited by the Lexical Entry Procedure. Passobj counterparts are PN and PNTHATSVO.

2.17 SNWH

Not currently implemented. This is an indirect question, an embedded clause beginning with a wh-word. Example: *I know who borrowed the car, She wondered whether it would snow.* Passobj counterparts are SNWH and NULLOBJ, as in *It was finally revealed who stole the car,* or *What he was really up to that day was revealed months later at the investigation.*

2.18 NSNWH

Not currently implemented. This is an NP object followed by indirect question, as in *He asked us whether it would snow.* Passobj counterparts: SNWH, NULLOBJ.

2.19 NTHATS

This is an NP followed by a THATS, as in *She told the factory that the sac was inoperative*:

```
OPS:   past
VERB:  tell
SUBJ:  pro:   she (sing)
D_OBJ: the factory (sing)
OBJ:   OPS:   past
       VERB:  be
       SUBJ:  the sac (sing)
       ADJ:   inoperative
```

Note that the NP object is marked as a dative object (semlabel `d_obj`, formerly `inner_obj`). This is because of the parallelism with dative constructions like *He told the factory the truth.*

Passobj counterpart: THATS. The semlabelling of this construction in passive is currently being refined in order to distinguish between cases like *He was told that the pump was inoperative*, where the subject should be marked as `d_obj`; and *It was said that the pump was defective*, where expletive *it* should not be represented in the argument structure at all.

2.20 SVEN

This is a predicative small clause, as in *He had the sac repaired quickly*:

```
OPS:   past
VERB:  have
SUBJ:  pro:   he (sing)
OBJ:   OPS:   untensed
       VERB:  repair
       SUBJ:  passive
       OBJ:   the sac (sing)
       ADV:   quickly
```

This sentence is ambiguous between SVEN and NSTGO analyses of the object: the NSTGO reading can be paraphrased *He had the sac which had been repaired quickly*, while the SVEN reading can be paraphrased *He caused the sac to be repaired quickly*. In the latter case, no one need be in possession of the sac. This difference is clearer still in *She found the book missing*. Clearly, *book* is not itself an argument of *find*, since the book was not found; what was found (out) was the proposition *the book is missing*. There's a lot of variation here, though: sometimes the subject of the small clause under *find* also seems to be an argument of the verb, especially in the passive (*The car was found parked on Elm Street*). Other verbs are clearer: *They reported the car stolen* doesn't mean that they reported the car, nor does *He had the stairs fixed* mean that he had the stairs. Probably one should split hairs and use two different BNF nodes corresponding to the NTOVO vs. OBJTOVO (exceptional case marking vs. object-controlled equi) distinction.

Passobj counterpart: VENPASS, as in *The gear teeth were found stripped and corroded.* SVEN doesn't always passivize, as above. (ISR rule is still under development for this passobj.)

2.21 NN

NN is the double object dative, as in *The factory found her a new pump* or *They told her the result*:

```
OPS:   past
VERB:  tell
SUBJ:  pro:   they (pl)
D_OBJ: pro:   her (sing)
OBJ:   the result (sing)
```

Note that the indirect object is semlabelled `d_obj`.

Passobj counterpart is NSTGO, as in *She was told the result*:

```
OPS:   past
VERB:  tell
SUBJ:  passive
D_OBJ: pro:   she (sing)
OBJ:   the result (sing)
```

Note that NP + NP sequences need not be parsed as NN. *I gave Ruth a good answer* contains NN, but *I consider Ruth a good dancer* is SOBJBE (below).

Many but not all NNs have counterparts with the *to-* or *for-* dative; thus *give books to Louise* alternates with *give Louise books*. However, in some cases only the prepositional form is found

(compare the meaning of *I got my degree for my parents (not for myself)* with that of *I got my parents my degree*); in other cases, we find only NN, as in *The book cost Mary five dollars.* The two constructions (NN and prepositional datives) have different semantic properties, so we do not want to attempt to represent them identically in the ISR.

2.22 SOBJBE

This is another small clause, consisting of subject followed by OBJBE (nstg, astg, or pn), as in *I consider him a genius* or *They consider it inoperative*:

```
OPS:    present
VERB:   consider
SUBJ:   pro:    they (pl)
OBJ:    OPS:    untensed
        VERB:   be
        SUBJ:   pro:    it (sing)
        ADJ:    inoperative
```

Sager has further subcategorization for nstg or astg or pn (or dstg, not included here) via **bvals** in the lexicon, since some verbs do not allow all OBJBE options; cf. *That made her angry, That made her the reigning monarch, *That made her in a state of rage.* PUNDIT's Lexical Entry Procedure does not currently elicit **bvals**.

The passobj counterpart is OBJBE, as in *He is considered a genius by his associates* or *It is considered inoperative*:

```
OPS:    present
VERB:   consider
SUBJ:   passive
OBJ:    OPS:    untensed
        VERB:   be
        SUBJ:   pro:    it (sing)
        ADJ:    inoperative
```

2.23 NA

This is a sequence of NP followed by an adjective phrase, as in *She painted the barn red* or *they stripped the gears bare*:

```
OPS:    past
VERB:   strip
SUBJ:   pro:    they (pl)
OBJ:    the gear (pl)
RES_CL:OPS:     untensed
        VERB:   be
        SUBJ:   the gear (pl)
        ADJ:    bare
```

The NA object type differs from SOBJBE in several respects. First, in NA the NP is an argument of the verb; if one paints the barn red, one has definitely painted the barn, whereas to have found the book missing is not to have found the book, and to believe the problem insoluble is not to believe the problem. Furthermore, the predication relationship between the adjective phrase and the NP is interpreted as a result in the case of NA. Finally, there is sometimes idiosyncratic selection between verb and adjective in NA, but not in SOBJBE. Thus *We sanded it smooth* sounds fine, but *We sanded it ugly* sounds odd, even if the ugliness is interpreted as resulting from the sanding.

The passobj counterpart is ASTG, as in *The house was painted red* or *It was stripped bare*:

```
OPS:    past
VERB:   strip
SUBJ:   passive
OBJ:    pro:    it (sing)
RES_CL:OPS:     untensed
       VERB:    be
       SUBJ:    pro:    it (sing)
       ADJ:     bare
```

2.24 ASTG

Example: *It went bad*:

```
OPS:    past
VERB:   go
SUBJ:   pro:    it (sing)
ADJ:    bad
```

Verbs with the ASTG object select for particular adjectives, as in *He went mad* (vs. the anomalous *He went sane*); and do not subcategorize for other OBJBE options (**He went a madman*). But it seems semi-semantic: *He turned blue/green/mean/sour/serious* but **He turned old/happy*. Thus it might not be possible to subcategorize for specific lexical items.

No passive.

2.25 DSTG

This is also quite rare. Certain verbs subcategorize for specific adverbs (*He means well* vs. **He means warmly*, or *She did beautifully* vs. **She did quietly*). No passive.

2.26 DP1

This is the simplest verb-particle combination, as in *He showed off*, *We lined up* (vs. **He showed out*, **We lined over*), or *Engine jacks over*:

```
OPS:   present
VERB:  jack
SUBJ:  engine (sing)
PTCL:  over
```

No passive.

2.27 DP2

DP2 is a particle followed by an NP, as in *He ran up the bill.* In contrast, *He ran up the hill* in its normal interpretation is NOT a DP2, but is rather a PN object. One test: only particles can occur to the right of the noun: *He ran the bill up* vs. **He ran the hill up*, to cite a classic example. Another test: only a PN can be topicalized, since it's a constituent: *Up the* HILL *he ran* vs. **Up the* BILL *he ran.* Another example: *They blew up the ship*:

```
OPS:   past
VERB:  blow
SUBJ:  pro:   they (pl)
PTCL:  up
OBJ:   the ship (sing)
```

Passobj counterpart is the particle, DP1, as in *A huge bill was run up that evening* or *The ship was blown up*:

```
OPS:   past
VERB:  blow
SUBJ:  passive
OBJ:   the ship (sing)
PTCL:  up
```

2.28 DP3

DP3 is just the permuted version of DP2, where the particle follows the noun phrase. Same passobj as DP2; order regularized in ISR. Since there are no transformations in PUNDIT, such alternations as that between DP2 and DP3 must be handled lexically.

2.29 DP1PN

This is a particle followed by a PP: *She moved in on him, They found out about it, The factory should have followed up on it*:

```
OPS:   past,shall,perf
VERB:  follow
SUBJ:  the factory (sing)
PTCL:  up
PP:    on
       pro:   it (sing)
```

Passobj counterpart is DP1P, when it passivizes, as in *The announcement was led up to by a series of remarks about the company's financial difficulties*(?), or *It should have been followed up on*:

```
OPS:   past,shall,perf
VERB:  follow
SUBJ:  passive
PP:    on
       pro:   it (sing)
PTCL:  up
```

2.30 DP2PN

DP2PN is a DP2 (particle + NP) followed by a PN, as in *He mixed up the apples with the pears.*

Passobj counterpart: DP1PN, as in *The apples were mixed up with the pears.* (Not, for example, **The pears were mixed up the apples with.*)

2.31 DP3PN

This is a DP3 (NP + particle) followed by a PN, as in *mix the apples up with the pears.* Passobj counterpart is also DP1PN.

2.32 DPSN

DPSN is a particle followed by a clause, as in *She found out where it was hidden, He pointed out that it was noon already, They often make out to be villains,* or *She found out that it was inoperative*:

```
OPS:   past
VERB:  find
SUBJ:  pro:   she (sing)
OBJ:   OPS:   past
       VERB:  be
       SUBJ:  pro:   it (sing)
       ADJ:   inoperative
PTCL:  out
```

Passobj counterparts are DPSN, as in *It was pointed out frequently that the plan could not succeed.* and DP1 *Where it was hidden was never really found out.* Both sound a little marginal, but might occur.